НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Веб-додаток для організації онлайн-черги з підтримкою можливості оцінювання якості послуг»

Виконала:
студентка IV курсу, групи КП-61
Корунська Анна Михайлівна _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.т.н., доцент,
Дідковська Марина Віталіївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2019 р.

ЗАВДАННЯ
на дипломний проєкт студенці

Корунській Анні Михайлівні

1. Тема проєкту «Веб-додаток для організації онлайн-черги з підтримкою можливості оцінювання якості послуг», керівник проєкту Заболотня Тетяна Миколаївна доцент кафедри ПЗКС, к.т.н., доцент, затверджені наказом по університету від «25» травня 2020 р. №1181-с.

2. Термін подання студентом проєкту «12» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - огляд існуючих програмних рішень;
   - обгрунтування вибору засобів розроблення;
   - огляд розробленого веб-додатку;
   - аналіз реалізації програмного забезпечення.

5. Перелік обов'язкового графічного матеріалу:
   - структура бази даних. ERD-діаграма (креслення);
   - взаємодія системи зі смарт-контрактом. Блок-схема алгоритму (креслення);
   - дерево проблем (плакат);

– структура модулів серверної та клієнтської частини (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	14.11.2019	
2.	Розроблення та узгодження технічного завдання	28.11.2019	
3.	Розроблення структури веб-додатку	15.12.2019	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2019	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2020	
6.	Підготовка матеріалів другого розділу дипломного проєкту	20.02.2020	
7.	Програмна реалізація веб-додатку	10.03.2020	
8.	Тестування веб-додатку	17.03.2020	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2020	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	11.04.2020	
11.	Підготовка графічної частини дипломного проєкту	21.04.2020	
12.	Оформлення документації дипломного проєкту	26.05.2020	

Студент Анна КОРУНСЬКА

Керівник проєкту Тетяна ЗАБОЛОТНЯ

АНОТАЦІЯ

Даний проєкт присвячений розширенню функціональних можливостей онлайн-черг. Розглянуто та проаналізовано недоліки вбудованого в такі системи механізму рейтингування, зокрема висвітлено проблеми довіри користувачів до відгуків та оцінок спеціалістів через можливість підробки або цензурування інформації. Для вирішення проблеми протидії шахрайству або підробленню даних, що зберігаються, в онлайн-чергах запропоновано використання технології блокчейн та смарт-контрактів.

Наведено основні функціональні характеристики розробленого веб-сервісу. Описано алгоритм роботи з коментарями та оцінками користувачів, а також алгоритм зберігання даних для верифікації їхньої цілісності в смарт-контракті мережі Ethereum. Зазначені подальші шляхи розвитку веб-додатку.

ABSTRACT

This work is dedicated to broadening the functionality of online-queue services. The drawbacks of the rating mechanism built into such systems are considered and analyzed. Possible issues of users' trust in the reviews and evaluations of specialists due to the possibility of counterfeiting or censoring of information are highlighted. Blockchain and smart contract technology are proposed for addressing the problem of fraudery or tampering with stored data.

The basic functional characteristics of the developed web application are provided. The algorithm of work with comments and user ratings is described, as well as the algorithm data integrity verification in the Ethereum smart contract. Further approaches to web application development are outlined.

ДП.045440-01-90. Веб-додаток для організації онлайн-черги з підтримкою можливості оцінювання якості послуг. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Веб-додаток для	5	
	організації онлайн-черги		
	з підтримкою можливості		
	оцінювання якості послуг.		
	Технічне завдання		
ДП.045440-03-81	Веб-додаток для	55	
	організації онлайн-черги		
	з підтримкою можливості		
	оцінювання якості послуг.		
	Пояснювальна записка		
ДП.045440-04-51	Веб-додаток для	4	
	організації онлайн-черги		
	з підтримкою можливості		
	оцінювання якості послуг.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Веб-додаток для	13	
	організації онлайн-черги		
	з підтримкою можливості		
	оцінювання якості послуг.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Веб-додаток для	1	
	організації онлайн-черги		
	з підтримкою можливості		
	оцінювання якості послуг.		
	Структура бази даних.		
	ERD-діаграма		
ДП.045440-07-99	Веб-додаток для	1	
	організації онлайн-черги		
	з підтримкою можливості		
	оцінювання якості послуг.		
	Взаємодія системи зі		
	смарт-котрактом.		
	Блок-схема алгоритму		
ДП.045440-08-98	Веб-додаток для	1	
	організації онлайн-черги		
	з підтримкою можливості		
	оцінювання якості послуг.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ ОРГАНІЗАЦІЇ ОНЛАЙН-ЧЕРГИ З ПІДТРИМКОЮ МОЖЛИВОСТІ ОЦІНЮВАННЯ ЯКОСТІ ПОСЛУГ

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проекту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анна КОРУНСЬКА

2019

ЗМІСТ

1. Найменування та галузь застосування…….3
2. Підстава для розроблення…….....3
3. Призначення розробки……...3
4. Вимоги до програмного продукту……3
5. Вимоги до проектної документації……...4
6. Етапи проєктування……...5
7. Порядок тестування розробки……...5

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-додаток для організації онлайн-черги з підтримкою можливості оцінювання якості послуг.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання для пошуку спеціалістів та запису в онлайн-чергу, а також для збереження даних про оцінки спеціалістів таким чином, який би забезпечив довіру користувачів до рейтингових даних, які надає сервіс.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Веб-додаток повинен забезпечувати такі основні функції:

1) Наявність ролей користувача та спеціаліста.
2) Реєстрація користувачів та спеціалістів.
3) Можливість пошуку спеціалістів за критеріями.
4) Можливість запису в онлайн-чергу до обраного спеціаліста.
5) Отримання інформації про спеціалістів та їхній рейтинг за допомогою публічного API.

Додаткові вимоги:

1) Зберігання даних про оцінки користувачів у спосіб, який би виключав їхнє навмисне спотворення чи видалення.

2) Можливість верифікування цілісності коментаря чи відгуку будь-яким користувачем сервісу.

Серверну частину рзробки виконати мовою програмування Python, а клієнтську – за допомогою бібліотеки React. Смарт-контракти реалізувати мовою програмування Solidity та опублікувати у тестовій блокчейн мережі Ethereum.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Структура бази даних. ERD-діаграма»;
   - «Взаємодія системи із смарт-контрактом. Блок-схема алгоритму».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи……..14.11.2019

Розроблення та узгодження технічного завдання……....28.11.2019

Розроблення структури веб-додатку…….....15.12.2019

Розроблення дизайну сторінок та графічних елементів……..03.02.2020

Програмна реалізація веб-додатку……....17.03.2020

Тестування веб-додатку…….....03.04.2020

Підготовка матеріалів текстової частини проєкту……..….28.04.2020

Підготовка матеріалів графічної частини проєкту……...…12.05.2020

Оформлення технічної документації проєкту……..…25.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ ОРГАНІЗАЦІЇ ОНЛАЙН-ЧЕРГИ З ПІДТРИМКОЮ МОЖЛИВОСТІ ОЦІНЮВАННЯ ЯКОСТІ ПОСЛУГ

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проекту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анна КОРУНСЬКА

2020

ЗМІСТ

СПИСОК ВИЗЧЕНЬ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ....4
ВСТУП....7
1. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ....8
1.1. Огляд проблеми, яка вирішується ПЗ....8
1.2. Аналіз існуючих рішень....9
1.3. Перелік вимог до функціональності програмної системи....11
2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБЛЕННЯ....15
2.1. Технології розроблення back end частини системи....15
2.2. Технології розроблення front end частини системи....18
2.3. Обґрунтування вибору застосування технології блокчейн....21
2.4. Висновки....27
3. ОГЛЯД РОЗРОБЛЕНОГО ВЕБ-ДОДАТКУ....28
3.1. Загальна структура системи....28
3.2. Структура бази даних....31
3.3. Організація взаємодії із смарт-контрактами....34
4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....37
4.1. Особливості реалізації серверної частини....37
4.2. Особливості реалізації клієнтської частини....41
4.3. Особливості реалізації смарт-контрактів....43
4.4. Дизайн та вміст веб-сторінок....46
4.5. Рекомендації щодо подальшого вдосконалення....55
ВИСНОВКИ....57
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ....59

СПИСОК ВИЗЧЕНЬ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Веб-сервіс – програмна система, що ідентифікується URI, і публічні інтерфейси та прив'язки якої визначені та описані мовою XML.

Веб-браузер – програмне забезпечення, що дає можливість взаємодіяти з елементами на гіпертекстовій веб-сторінці.

ПЗ – програмне забезпечення.

БД – база даних.

Back-end – частина сайту, невидима відвідувачеві, що безпосередньо відповідає за роботу першого.

Front-end – лицьова частина сайту, яку бачить та з якою безпосередньо взаємодіє відвідувач.

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією.

JavaScript – динамічна, об'єктно-орієнтована прототипна мова програмування.

Фреймворк – програмне забезпечення, яке полегшує процес розроблення на об’єднання різних модулів програмного проекту.

Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript.

RESTful – Representational State Transfer, підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів.

WSGI – Web Server Gateway Interface – стандарт взаємодії між Python-програмою, яка виконується на стороні сервера, і самим веб-сервером, наприклад, Apache.

Unicode – Юнікод, Уніфіковане Кодування – промисловий стандарт, розроблений, щоб забезпечити цифрове представлення символів усіх писемностей світу та спеціальних символів.

React – React.js, ReactJS – відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків.

Angular – написаний на TypeScript front-end фреймворк з відкритим кодом, який призначений для побудови динамічних клієнтських веб-інтерфейсів.

DOM – Document Object Model, Об'єктна модель документа – специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами (як правило, документами XML).

XML – Extensible Markup Language, Розшиирювана мова розмітки – запропонований консорціумом World Wide Web Consortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет.

HTML – HyperText Markup Language, мова розмітки гіпертексту – це мова тегів, якою пишуться гіпертекстові документи для мережі Інтернет.

AJAX – Asynchronous JavaScript And XML – підхід до побудови користувацьких інтерфейсів веб-застосунків, за яких вебсторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер і сама звідти довантажує потрібні користувачу дані.

Blockchain – розподілена база даних, що зберігає впорядкований ланцюжок записів (так званих блоків), що постійно довшає; дані захищено від підробки та спотворення, а кожен блок містить часову позначку, хеш попереднього блока та дані транзакцій, подані як хеш-дерево.

Ethereum – крипто-валюта та платформа для створення децентралізованих онлайн-сервісів (dapps) на базі блокчейна, що працюють на базі розумних контрактів.

EVM – Ethereum Virtual Machine – віртуальна машина та середовище виконання смарт-контрактів, що опубліковані в мережі Ethereum.

Block Explorer – веб-сайт або інструмент, який дозволяє переглядати блоки, переглядати адреси гаманців, мережевий хешрейт, дані транзакцій і іншу ключову інформацію в блокчейн-мережі.

Etherscan – веб-додаток, що слугує block explorer сервісом для основних та тестових мереж блокчейну Ethereum.

HTTP – Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів – протокол передачі даних, що використовується в комп'ютерних мережах.

ВСТУП

Велика кількість закладів та установ за своїм принципом роботи передбачають необхідність почергового обслуговування відвідувачів чи користувачів послуг. Найчастіше це спричиняє необхідність очікування і формування черг бажаючих скористатися послугами. Так звані “живі” черги мають велику кількість недоліків, зокрема те, що на перебування в них люди можуть витратити багато часу, а крім того, такі черги погано піддаються управлінню. Автоматизація та систематизація цього процесу за допомогою сервісів онлайн-черг допомагає вирішувати згадані проблеми.

Системи онлай-черг дозволяють знаходити потрібних спеціалістів, проте у виборі користувачі зазвичай керуються відгуками попередніх клієнтів спеціаліста чи послуги. Проблема полягає в тому, що відгуки та оцінки про роботу спеціалістів можуть не викликати довіри через людьский фактор, можливість підкупу, цензурування тощо.

Також, існуючі аналоги серед систем онлайн-черг, такі як doc.ua, HELSI або електронна черга серсісного центру МВС України є сильно орієнтованими лише не один контретний вид послуг, що не є універсальним сервісом, яким користувачі могли б керуватися в більшості ситуацій при пошуку спеціалістів.

Таким чином, розроблення універсального сервісу онлайн-черги, який би забезпечив чесне оцінювання спеціалістів чи якості послуг та запобігав би при цьому видаленню та цензуруванню відгуків, є актуальною задачею.

Об’єктом дослідження даного проєкту є функціонування системи онлайн-черги з точки зору її захисту від шахрайства при побудові рейтингу спеціалістів чи оцінці якості наданих послуг.

1. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

1.1. Огляд проблеми, яка вирішується ПЗ

Онлайн-чергою або електронною чергою називається програмна система, що надає можливість систематизації та формалізації потоку відвідувачів в приватному або державному сервісі [1]. Крім власне функціональних можливостей щодо запису користувачів до черги, сучасні програмні системи такого типу підтримують широкий спектр супровідних можливостей: облік та оцінка результативності конкретних спеціалістів, отримання статистики про послуги, що користуються найбільшим попитом, відстеження динаміки звернень. Велика кількість подібних систем надає також можливість переглядати оцінки та відгуки про роботу певного спеціаліста, побудову рейтингів тощо.

Не дивлячись на загальновживаність систем з підтримкою рейтингування та впорядкування взаємодії з профільними спеціалістами, достовірність інформації, котру вони надають, не завжди є задовільною для користувачів. Це впливає на рівень довіри користувачів до сервісу, а також на те, наскільки їм легко знаходити потрібних спеціалістів online.

Основна проблематика подібних систем полягає у тому, що відгуки та оцінки не викликають довіри. Однією з причин є те, що негативні відгуки або оцінки після публікації можуть бути відредаговані або видалені автором або сервісом, де було опубліковано відгук. Інша причина – схильність користувачів залишати переважно негативні відгуки.

Крім того, сервіси онлайн-запису до спеціалістів не завжди можуть надавати користувачам вичерпну та достовірну інформацію про освітні та професіональні здобутки спеціалістів, детальну інформацію про їх досвід та спеціалізацію.

Дерево проблем наведено на рис. 1.1.

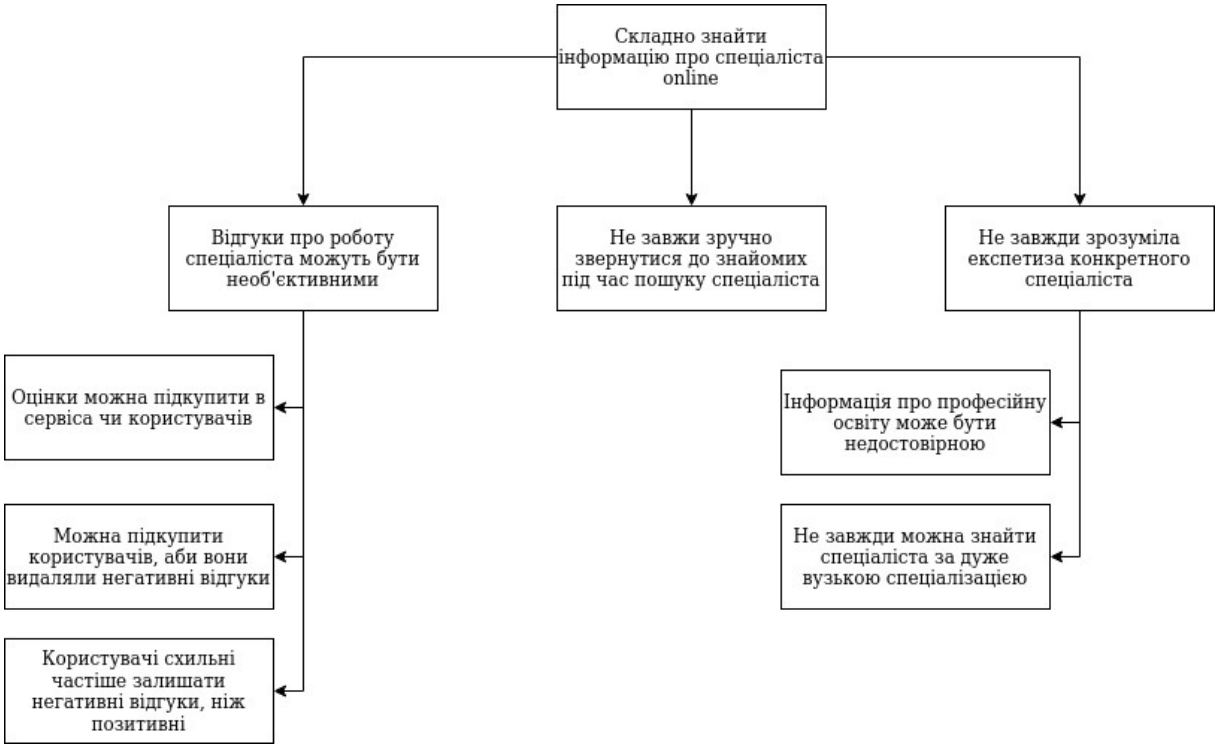


Рис. 1.1. Дерево проблем

Таким чином, метою розроблюваного ПЗ є вирішення проблеми довіри користувачів до відгуків про роботу спеціалістів шляхом побудови онлайн-черги з підтримкою прозорого рейтингування послуг.

1.2. Аналіз існуючих рішень

Під час аналізу існуючих рішень було знайдено декілька веб-сервісів, котрі надають можливість запису до спеціалістів, дозволяють переглядати їхній рейтинг та відгуки про роботу або полегшують зворотній зв’язок з ними.

При аналізі було знайдено сервіси онлайн-черг, що слугують для формалізації та запису до черг в державних закладах та уставовах. Наприклад, сервіс HELSI використовується для запису на прийом до лікарів в державних закладах охорони здоров’я, а сервіс електронної черги

сервісного центру МВС України дозволяє записуватись онлайн на такі послуги, як оформлення транспортних засобів. Недолік цих сервісів полягає у тому, що вони сконцентровані на державних послугах і призначені лише для одного напряму послуг. Крім того, в таких системах не завжди проводиться оцінювання і дані про рейтинг послуг не завжи є доступною інформацією для користувачів.

Веб-сервіс doc.ua [2] є найбільш близьким зі знайдених аналогів до розроблюваного. Даний ресурс дозволяє знаходити лікарів різноманітних спеціальностей у певному місці та записуватися до них на прийом.

Doc.ua дозволяє переглядати рейтинг та відгуки про лікаря. Відгуки можуть залишати лише ті користувачі, котрі записалися на прийом через систему – таким чином підвищується об'єктивність відгуків.

Приклад відображення даних про лікаря (ім'я, спеціальність, стаж, рейтинг, кількість відгуків, місце на адреса роботи, ціна за прийом) можна побачити на рис. 1.2.

Рис. 1.2. Відображення даних про лікаря в сервісі doc.ua

Доступна доволі детальна інформація про спеціаліста та його діяльність: освіта, участь у семінарах, досягнення, досвід та пріоритетні напрями роботи, хвороби, що є спеціалізацією даного лікаря тощо.

Користувач може обрати зручний день та час і записатися на прийом, інтерфейс для вибору часу наведено на рис 1.3. При цьому потрібно вводити свої контрактні дані.

Варто також відзначити те, що при визначенні рейтингу враховується не тільки середнє арифметичне між оцінками користувачів. На нього також впливатимуть стаж роботи, освітній рівень та науковий ступінь.

Серед основних недоліків doc.ua можемо визначити сильну сконцентрованість на медичній тематиці, а також той факт, що негативні відгуки про певного лікаря все ще можуть цензуруватися самим сервісом.

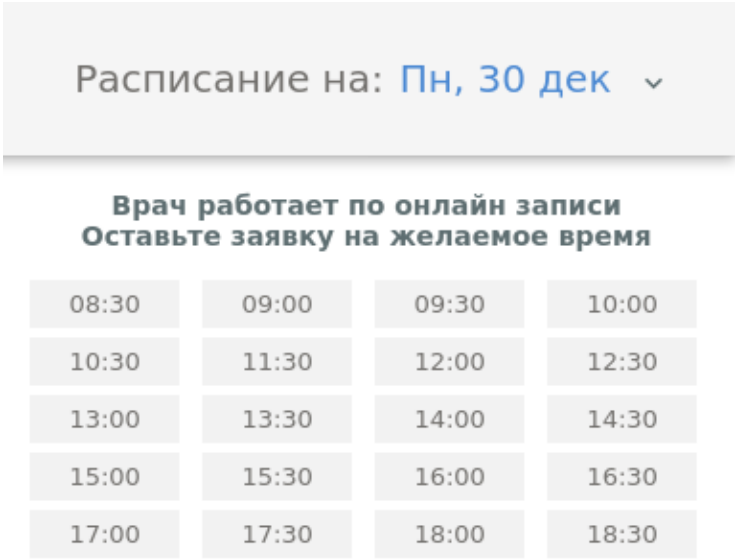


Рис 1.3. Форма запису до лікаря: вибір часу

Крім того, певні аспекти реалізації користувацького інтерфейсу не є зручними, як то відсутність можливості пошуку за проміжком часу у певному районі міста.

1.3. Перелік вимог до функціональності програмної системи

Проаналізувавши сферу сервісів онлайн-черг та існуючі веб-сервіси, що надають послуги упорядкування записів до спеціалістів та їх

рейтингування, а також беручи до уваги дерево проблем (рис. 1.1.), було визначено цілі та бажані результати проєкту у вигляді дерев цілей та результатів відповідно, а також сформульовано вимоги до розроблюваної системи.

На рис 1.4. зображено дерево цілей проєкту:

Рис.1.4. Дерево цілей проєкту

На рис 1.5. наведено дерево результатів проєкту.

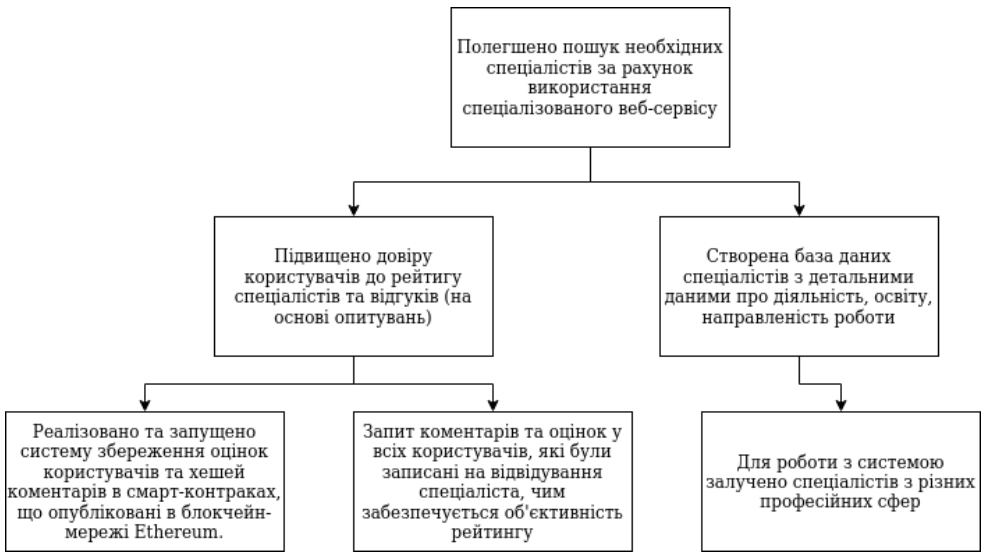


Рис. 1.5 Дерево результатів проєкту

З огляду на вищезазначене, можна виділити такі функціональні вимоги до розроблюваного сервісу:

1. Система має надавати можливість реєстрації для спеціалістів різних сфер діяльності з можливістю заповнювати професійні персональні дані.
2. Система має надавати можливість спеціалісту обирати місто та адресу прийомів, встановлювати графік прийомів.
3. Система має надавати можливість спеціалісту переглядати усіх користувачів, які записані до нього, в форматі розкладу з основними контактними даними.
4. Система має надавати можливість користувачам зареєструватися та створити свій особистий кабінет.
5. Система має дозволяти користувачам переглядати дані минулих та майбутніх записів у особистому кабінеті, а також скасовувати їх, отримувати контактні дані спеціаліста, тощо
6. Система має дозволяти користувачам записуватися на прийом на зручний для них час.
7. Система має дозволяти користувачам переглядати дані минулих та майбутніх записів у особистому кабінеті, а також скасовувати їх, отримувати контактні дані спеціаліста тощо.
8. Система має дозволяти тим користувачам, які записалися на прийом і відвідали його (тобто не відбулося скасування візиту), залишати оцінки та відгуки.
9. Оцінки та відгуки мають зберігатися системою в такому вигляді, щоб будь-який бажаючий міг пересвідчитися у тому, что коментарі або оцінки не були змінені або вилучені.
10. Будь який користувач має мати можливість верифікувати будь-який коментар на те, чи не був він підроблений.

На основі цих вимог було сформульовано такі нефункціональні вимоги:

1. Для функціонування системи мають бути створені такі ролі користувачів: звичайний користувач (клієнт) та спеціаліст.
2. Система має мати певну кількість категорій, в рамках яких можуть працювати спеціалістів, та підтримувати можливість легко розширювати їх кількість.
3. Тільки зареєстровані користувачі мають мати можливість записуватися на прийом.
4. Персональні контактні дані користувачів можуть бути доступні виключно тим спеціалістам, до котрих вони записалися на прийом і лише за певну кількість часу до та після прийому. Реєстрація на прийом може відбутися тільки в тому разі, що користувач надасть згоду на передачу його контактних даних спеціалісту.
5. Для зберігання даних про оцінки та відгуки про спеціалістів має бути використана технологія, яка дозволяє гаратнтувати цілісність та незмінність даних після збереження.
6. Зібрана статистика має бути доступна через публічний API.
7. Інтерфейс веб-сервісу має відображатись успішно і на мобільних пристроях, і на ПК.
8. Веб-сервіс має бути доступним у веб-браузерах Google Chrome, Mozilla Firefox, Safari.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБЛЕННЯ

2.1. Технології розроблення back end частини системи

Відповідно до поставленої задачі, а саме розроблення серверної частини сервісу онлайн-черги, виникає необхідність вибору оптимальних засобів розроблення.

Проведемо аналіз популярних засобів розроблення серверної частини, а саме мови програмування Python і фреймворку Python Flask і програмної платформи Node.js і фреймворку Express.

Проведемо аналіз засобів розроблення серверної частини, а саме мови програмування Python і фреймворку Python Flask і програмної платформи Node.js і фреймворку Express, оскільки вони є найбільш розсповлюдженими наборами інструментів для швидкої та ефективної серверної розробки та прототипування.

2.1.1. Мова програмування Python

Python – об'єктно-орієнтована інтерпретована мова програмування загального призначення високого рівня, що орієнтована на підвищення продуктивності розробника і читання коду [3]. Має сувору динамічну типізацію. Також підтримує роботу з високорівневими структурами даних, що дозволяють забезпечити швидкий процес розроблення програм. Крім того, можна відзначити мінімалістичний синтаксис ядра Python. У той же час стандартна бібліотека включає великий обсяг корисних функцій. Завдяки підтримці модулів та пакетів модулів Python дозволяє повторно використовувати код. Стандартні бібліотеки та інтерпретатор даної мови є доступними як у скомпільованій формі, так і у вихідній на всіх основних платформах. Python підтримує багато різноманітних парадигм програмування, зокрема: об'єктно-орієнтоване, функціональне, процедурне, імперативне і аспектно-орієнтоване програмування. Серед основних архітектурних рис можна відзначити динамічну типізацію, повну інтроспекцію часу виконання, автоматичне керування пам'яттю, підтримку

багатопоточних обчислень, механізм обробки виключень, а також високорівневі структури даних. Підтримується розбиття програм на модулі, які, в свою чергу, можуть об'єднуватися в пакети. Крім того, є інтерактивний режим роботи [3].

Мова Python має такі переваги як:

- крос-платформність – мова працює майже на всіх відомих платформах, таких як Microsoft Windows, всі варіанти UNIX, Mac OS, iPhone OS, Palm OS, Amiga та Android;
- велика кількість модулів, що дають можливості для роботи з мережевими протоколами, серверами та клієнтами, крос-платформними застосунками і графічним інтерфейсом;
- легка інтеграція і простий синтаксис.

Недоліки мови Python:

- невисока швидкість виконання програм;
- відсутність статичної типізації;
- помилки, що не виявляються на етапі компіляції.

2.1.2. Фреймворк Flask

Flask – фреймворк для створення веб-додатків мовою програмування Python [4]. В його основі лежить інструментарій Werkzeug, а також шаблонізатор Jinja2. Даний фреймворк відноситься до категорії мікрофреймворків, тобто мінімалістичних каркасів для веб-додатків, оскільки не вимагає спеціальних засобів чи бібліотек.

Фреймворк Flask не вводить додаткові рівні абстракції для роботи з веб-формами, базою даних та іншими компонентами веб-додатку, які надають широко розповсюджені функції за допомогою сторонніх бібліотек. Проте він має підтримку розширень, що забезпечують додаткові властивості, наче вони були доступні у фреймворку від самого початку. Таким чином, даний фреймворк компенсує звужену функціональність порівняно з full-stack фреймворками. Існують розширення для

встановлення об'єктно-реляційних зв'язків, перевірки форм, контролю процесу завантаження, підтримки різноманітних відкритих технологій аутентифікації та декількох поширених засобів для фреймворку [4].

Перевагами даного фреймворку є:

- наявність сервера для розробки та відлагоджувача;
- управління запитами RESTful;
- вбудована підтримка юніт-тестів;
- підтримка безпечних куків (сесії на стороні клієнта);
- 100% відповідність WSGI 1.0;
- докладна документація;
- захист від типових методів зламу;
- підтримка Unicode;
- наявність розширень для забезпечення бажаної поведінки;
- сумісність з Google App Engine.

Основним недоліком даного фреймворку є погана підтримка модульності.

2.1.3. Програмна платформа Node.js

Node.js – програмна платформа, що перетворює мову програмування JavaScript із вузькоспеціалізованої браузерної мови у повноцінну мову програмування широкого призначення, оскільки дозволяє виконувати функції мови JavaScript на сервері та надсилати користувачу результат їх виконання [5]. Вона застосовується для виконання високопродуктивних мережевих застосунків та створення серверних та клієнтських програм.

Дана платформа дозволяє підключати зовнішні бібліотеки, написані іншими мовами програмування, взаємодіяти з пристроями вводу та виводу. Переважно Node.js використовується в ролі веб-сервера, проте вона дозволяє також розробляти віконні десктопні додатки та програмувати мікроконтролери. В основі даної платформи полягає подійно-орієнтоване та асинхронне програмування.

Node.js використовує систему модулів CommonJS та рушій JavaScript Google V8. Для забезпечення обробки великої кількості паралельних запитів використовується асинхронна модель запуску коду, заснована на визначенні обробників зворотних викликів та обробці подій у неблокуючому режимі.

Стандартна комплектація Node.js включає в себе багато модулів, у яких реалізовані типові основні операції для взаємодії з операційною системою, системою файлів, мережею, протоколами, а такою утилітами для обробки даних.

2.1.4. Фреймворк Express.js

Express.js – мінімалістичний та гнучкий фреймворк для створення веб-додатків на Node.js, що реалізований я вільне відкрите програмне забезпечення під ліцензією MIT [6].

Серед його особливостей варто відзначити мінімалістичність та наявність великої кількості під'єднуваних модулів. Даний фреймворк забезпечує набір фундаментальних функції для веб-додатків та мобільних додатків, а також створення API, дає ряд готових абстракцій, які спрощують створення сервера та серверної логіки, а саме:

- обробка відправлених форм;
- робота з cookies.

Після проведеного аналізу для розроблення сервеної частини було обрано мову програмування Python та фреймворк Python Flask, оскільки ці технології, порівняно з Node.js та Express.js надають більш прості в використанні інструменти, що позитивно впливає та якість та швидкість розроблення ПЗ.

2.2. Технології розроблення front end частини системи

Відповідно до поставленої задачі, а саме розроблення клієнтської частини сервісу онлайн-черги, виникає необхідність вибору оптимальних засобів розроблення.

Проведемо аналіз популярних засобів розроблення клієнтської частини, а саме фреймворків React та AngularJS.

2.2.1. React

React або ReactJS – декларативний, ефективний і гнучкий фреймворк, написана мовою JavaScript, що використовується для розробки користувацьких інтерфейсів, а також односторінкових і мобільних додатків. Він надає високу швидкість розроблення, простоту і масштабованість. Також React дозволяє працювати з іншими бібліотеками.

Серед особливостей даного фреймворку [7]:

- Вони мають незмінний стан, що не може бути змінео напряму, цей механізм виконується через callback-функції;
- віртуальний DOM, що створюється завдяки кеш-структурі у пам’яті і забезпечує оптимальне оновлення DOM-браузера;
- використання JavaScript XML (JSX) – розширення синтаксису JavaScript, що дозволяє використовувати HTML-подібний синтаксис для опису структури інтерфейсу;
- наявність методів життєвого циклу, що дозволяють розробнику запускати код на різних стадіях життєвого циклу компонента;
- React Hooks, що дозволяють використовувати можливості React без створення класів. Також з їх допомогою можна розміщувати логіку компонента у повторно використовуваних функціях.

Серед переваг роботи з бібліотекою React варто відзначити:

- зрозумілість візуального представлення компонента через вихідний код;
- простота і читабельність коду.

Серед недоліків даної бібліотеки можна перелічити таке:

- відсутність системи подій;
- відсутність можливості роботи з AJAX.

2.2.2. AngularJS

AngularJS – фреймворк для мови JavaScript з відкритим вихідним кодом [8]. Призначений для розроблення односторінкових веб-додатків, які він розширює за допомогою шаблону MVC, а також спрощує тестування і розробку. Даний фреймворк працює з HTML, що містить додаткові користувацькі атрибути, які мають назву директива, і пов'язує ввід або вивід області сторінки з моделлю, що являє собою звичайні змінні мови JavaScript. Значення цих змінних задається вручну або підтягується із стратичних або динамічних JSON-даних.

AngularJS спроектовано відповідно до логіки декларативного програмування, що найкраще підходить для побудови користувацьких інтерфейсів та опису програмних компонентів, а також для імперативного програмування, що найкраще підходить для опису бізнес-логіки. Даний фреймворк розширює та адаптує класичну розмітку HTML, забезпечуючи таким чином двосторонню прив'язку даних для динамічного контенту. Це дозволяє автоматично синхронізувати модель і представлення. Таким чином, AngularJS зменшує роль DOM-маніпуляцій і покращує тестування.

Серед переваг даного фреймворку варто зазначити:

- розділення клієнтської та серверної частини коду, що дозволяє вести паралельну розробку;
- відділення DOM-маніпуляцій від логіки веб-додатку.

AngularJS дотримується MVC-шаблону проектування і підтримує слабкий зв'язок між представленням, даними та логікою компонентів. Використовуючи впровадження залежності, даний фреймворк переносить на клієнтську сторону такі класичні серверні служби, як видозалежні контролери. Відповідно, зменшується навантаження на сервер, що робить веб-додаток більш легким.

Двостороннє зв'язування даних у фреймворку AngularJS є однією з найбільш важливих особливостей, що дозволяє зменшувати кількість коду, звільняючи сервер від роботи із шаблонами. Замість цього шаблони

відображаються в якості звичайного HTML, що містить дані області, визначеної у моделі. Спеціальний сервіс стежить за змінами в моделі та змінює розділ HTML-виразу у представленні через контролер. Крім того, будь-які зміни представлення відображаються у моделі. Це дозволяє обійти необхідність маніпулювати DOM і полегшує ініціалізацію і прототипування веб-додатку.

Після аналізу було вирішено для розроблення клієнтської частини обрати бібліотеку React, оскільки вона підтримує використання JSX та система React Hooks є зручною для перевикористання компонентів системи.

2.3. Обґрунтування вибору застосування технології блокчейн

Технологія блокчейн отримала широке розповсюдження та привернула до себе увагу в контексті криптовалют, проте, згодом, вона отримала подальше застосування в інших галузях. Підвищена увага до технології пов’язана з потенціалом підходу, в котрому кінцевий стан бази даних певної системи або мережі криптографічно захищений від несанкціонованих змін. Таким чином, в системах з великою кількістю незалежних сторін, блокчейн дозволив забезпечити безпечну синхронізацію даних та, з деякими допущеннями, їхню незмінність.

В сенсі такого підходу блокчейн можна визначити як спосіб зберігання та синхронізації даних між певною групою учасників, який не вимагає взаємної довіри. Блокчейн припускає зберігання даних згрупованими в блоки, кожен з яких криптографічно пов’язаний з попереднім [9].

Перевірка цілісності даних, що зберігається, забезпечується завдяки тому, що кожен блок містить у собі значення хеша попереднього блока. Спроба змінити дані в певному блоці призведе до того, що його хеш значення зміниться. Відповідно, ланцюжок не буде неперервним і будь-який учасник системи зможе побачити, що дані блоку були пошкоджені.

Така особливість робить практично неможливим переписування даних, які вже було включено до блоку.

Втім, теоретично, існує можливість зміни даних, що було включено до ланцюжка. Змінити дані вже існуючого ланцюга дійсно неможливо, проте теоретично можна створити альтернативний ланцюжок і зробити його основним. Для цього треба переконати більшість мережі використовувати альтернативний ланцюжок для того, щоб створювати нові блоки на його основі. Зазвичай, таке можливо, якщо альтернативний ланцюжок буде довшим за основний та відповідатиме всім вимогам протоколу. Атака такого роду називається атакою 51% і є можливою тоді, коли більша частина мережі змовиться про "переписування" даних. В великих розподілених системах, як Bitcoin або Ethereum, таке є практично неможливим, оскільки усі їхні учасники анонімні і сильно розподілені географічно [10].

Можна виділити такі суттєві аспекти побудови програмного забезпечення, котрих технологія блокчейн допомагає досягти [9]:

1. Цілісність історії змін бази даних.
2. Мінімізація затримок синхронізації та резервного копіювання.
3. Підтримка валідування даних з боку групи включених користувачів.
4. Можливість проведення аудиту в режимі реального часу.
5. Фіксація даних, що включаються до бази даних, у часі.
6. Trustless, тобто необхідний рівень довіри користувачів до системи мінімальній (система побудована таким чином, що мінімізує можливість обману, підробок чи інших негативних впливів людського фактору).

Серед недоліків використання технології блокчейн можна зазначити такі:

1. Використання технології блокчейн передбачає підвищену відповідальність з боку користувачів.

2. Безпека та недоторканність інформації або криптовалюти залежить від цілісності та захищеності приватних ключів користувача.
3. Використання технології блокчейн може негативно вплинути на швидкодію та ефективність роботи ПЗ.
4. У багатьох основних публічних мережах криптовалют виконання транзакцій передбачає оплату комісій.

Перераховані недоліки є доволі суттєвими і тому в багатьох типах ПЗ використання технології блокчейн не є доцільним рішенням. Втім, існує низка класичних способів застосування технології блокчейн, серед яких можемо виділити:

- системи голосування;
- системи забезбечення довіри;
- зберігання медичної інформації;
- публічні реєстри;
- ланцюжки поставок;
- страхування.

Для кожного з цих сценаріїв застовування ефективність впровадження технології блокчейн є доволі високою.

Виходячи з цього, технологія блокчейн якнайкраще підходить для протидії шахрайству з видаленням даних або цензуруванню. Ця властивість є неймовірно важливою для того, аби забезпечити довіру до розроблюваної системи.

Розглянемо існструменти, які дозволяють використовувати переваги технології блокчейн у програмному забезпеченні, а саме середовище виконання та мову написання смарт-контрактів, а також фреймворк тестування та публікації смарт-контрактів.

2.3.1. Мережа Ethereum

Ethereum – це відкрита, загальнодоступна розподілена обчислювальна платформа на основі технології блокчейн з підтримкою

роботи смарт-контрактів. Вона підтримує модифіковану версію консенсусу Накамото через переходи на основі транзакцій [8]. Ethereum друга мережа за величиною по капіталізації ринку після Bitcoin [9]. Валютою мережі Ethereum є Ether (ETH).

Ethereum надає децентралізовану віртуальну машину, віртуальну машину Ethereum (EVM),іртуальну машину Ethereum (EVM), яка може виконувати смарт-контракти за допомогою міжнародної мережі публічних вузлів. Відмінність від інших платформ, таких як Bitcoin з Bitcoin Script, полягає у тому, що EVM дозволяє виконання інструкцій смарт-контрактів, написані Тюрінг-повною мовою.

При запуску на блокчейн смарт-контракт стає схожим на самостійну комп'ютерну програму, яка автоматично яка автоматично виконується за настання певних умов. На блокчейні смарт-контракти дозволяють коду працювати точно, як запрограмовано, без будь-якої можливості простою, цензури, шахрайства або втручання третіх сторін. Смарт-контракти можуть сприяти обміну грошей, змісту, власності, акцій або будь-яких інших цінностей.

Для написання смарт-контрактів на Ethereum розроблені такі мови програмування як Solidity, Serpent, LLL, Mutan, Vyper. З них Serpent та Mutan вважаються застарілими та не підтримуються.

Gas або «газ», внутрішній механізм ціноутворення транзакцій, використовується для пом’якшення спаму та розподілу ресурсів у мережі.

Для захисту від спаму та розподілу ресурсу мережі використовується gas, внутрішній механізм ціноутворення. За кожну транзакцію, що має бути оброблена майнерами, користувач має сплатити певну кількість газу, який автоматично буде придбано за ETH. Таким чином, користувач оплачує ресурси повних вузлів, які підуть на обробку його операції.

Інтерес до мережі виявляли як великі компанії (Microsoft, IBM, Acronis, банківський консорціум R3), так і невеликі бізнеси та стартапи.

2.3.2. Мова програмування Solidity

Solidity – об'єктно-орієнтована, предметно-орієнтована мова програмування [10], була запропонована в серпні 2014 року Гевіном Вудом і розроблена в рамках роботи над проектом Ethereum. Мова спроектована для трансляції в байт код віртуальної машини Ethereum. Широко використовується при створенні смарт-контрактів та децентралізованих застосунків.

Solidity використовується для написання самовиконуваних контрактів для платформи Ethereum, виконання контрактів виконується на EVM. Дозволяє розробникам створювати самодостатні додатки, що містять бізнес-логіку, що призводить до виконання накладених умов і зобов'язань та їхню незмінність.

Після написання контракт має бути трансльовано в байт-код для EVM та опубліковано в мережі Ethereum. Процес публікації контракту передбачає надсилання транзакції із байт-кодом контракту до мережі від імені його розробника. При цьому користувач, що надсилає транзакцію, має заплатити за публікацію контракту за допомогою gas. Контракт може бути опубліковано в основній мережі Ethereum або в одній з тестових мереж: Ropsten network, Kovan network, Rinkeby network.

Перевагами даної мови є:

- докладна документація;
- статична типізація;
- використання синтаксису ECMAScript;
- підтримка комплексних змінних контрактів, включаючи довільні ієрархічні відображення (mappings) і структури;
- підтримка наслідування контрактів, в тому числі множинного наслідування;
- наявність бібліотек готових рішень (наприклад Open Zeppelin Contracts [11]);

- зручні інструменти розробки та тестування смарт контрактів – онлайн IDE Remix [12], фреймворк розробки та тестування контрактів Truffle, тощо.

Недоліками мови Solidity є:

- особливість у використанні відображень (mappings), через яку не можна відобразити усі записані пари ключ-значення;
- неможливість повернення масивів даних із функцій;
- відсутність підтримки безпечної нецілочисельної арифметики
- неможливість видалення або модифікації контракту після його публікації, якщо така можливість не була передбачена завчасно.

Для розроблення смарт-контрактів оберемо мову програмування Solidity, оскільки вона є добре підтримуваною та документованою на відміну від Serpent та Mutan, що вважаються застарілими. Також, Solidity вигідно відрізняється від LLL та Vyper тим, що це високорівнева мова програмування, що часто використовується у великих проєктах.

2.3.3. Використання Truffle suite для тестування смарт-контрактів

Truffle – середовище розроблення та тестування смарт-контрактів, яке сумісне з блокчейнами, які виконують смарт-контакти на EVM [16]. Truffle використовує фреймворки Mocha та Chai для тестування і створює обгортки над ними, що адаптовані для зручності тестування смарт-контрактів.

Фреймворк підтримує інтерграцію з Ganache та Drizzle, які пришвидшують розробку децентралізованих додатків. Ganache – інструмент для розгортання приватного блокчейну з одного вузла, що пришвидшує процес розробки та тестування за рахунок простоти користування приватною мережею та більшій швидкості відповіді від мережі. Drizzle це біблітокека для розроблення типових користувацьких веб-інтерфейсів, що орієнтовані на децентралізовані додатки та взаємодію із мережею блокчейн.

Перевагами даного фреймворку є [16]:

- автотестування смарт-контрактів;
- автоматичне розгортання смарт-контрактів в приватній блокчейн-мережі Ganache [17] під час виконання тестів;
- підтримка компіляції, лінкування та пубкікації смарт-контрактів;
- підтримка скриптів для публікації та мігрування смарт-контрактів;
- менеджмент мережевих налаштувань для публікації контрактів в будь-яку кількість публічних або приватних блокчейн-мереж;
- інтерактивна консоль для прямої взаємодії із смарт-контрактами;

2.4. Висновки

Після проведеного аналізу інструментів розроблення для реалізації серверної частини було обрано мову програмування Python та фреймворк Python Flask.

Для розроблення клієнтської частини було обрано бібліотеку React.

Для інтеграції з веб-додатком було обрано мережу Ethereum із смарт-контрактами мовою Solidity, а також Truffle Suite для тестування та організації процесу публікування смарт-контрактів до тестової мережі Ethereum.

3. ОГЛЯД РОЗРОБЛЕНОГО ВЕБ-ДОДАТКУ

3.1. Загальна структура системи

Програмна система реалізована у вигляді веб-додатку. Її можна розділити на три суттєві частини: серверна (back-end), клієнтська (front-end), смарт-контракт, що опубліковано в мережі Ethereum.

Узагальнена структурна організація веб-додатку представлена на рис. 3.1.

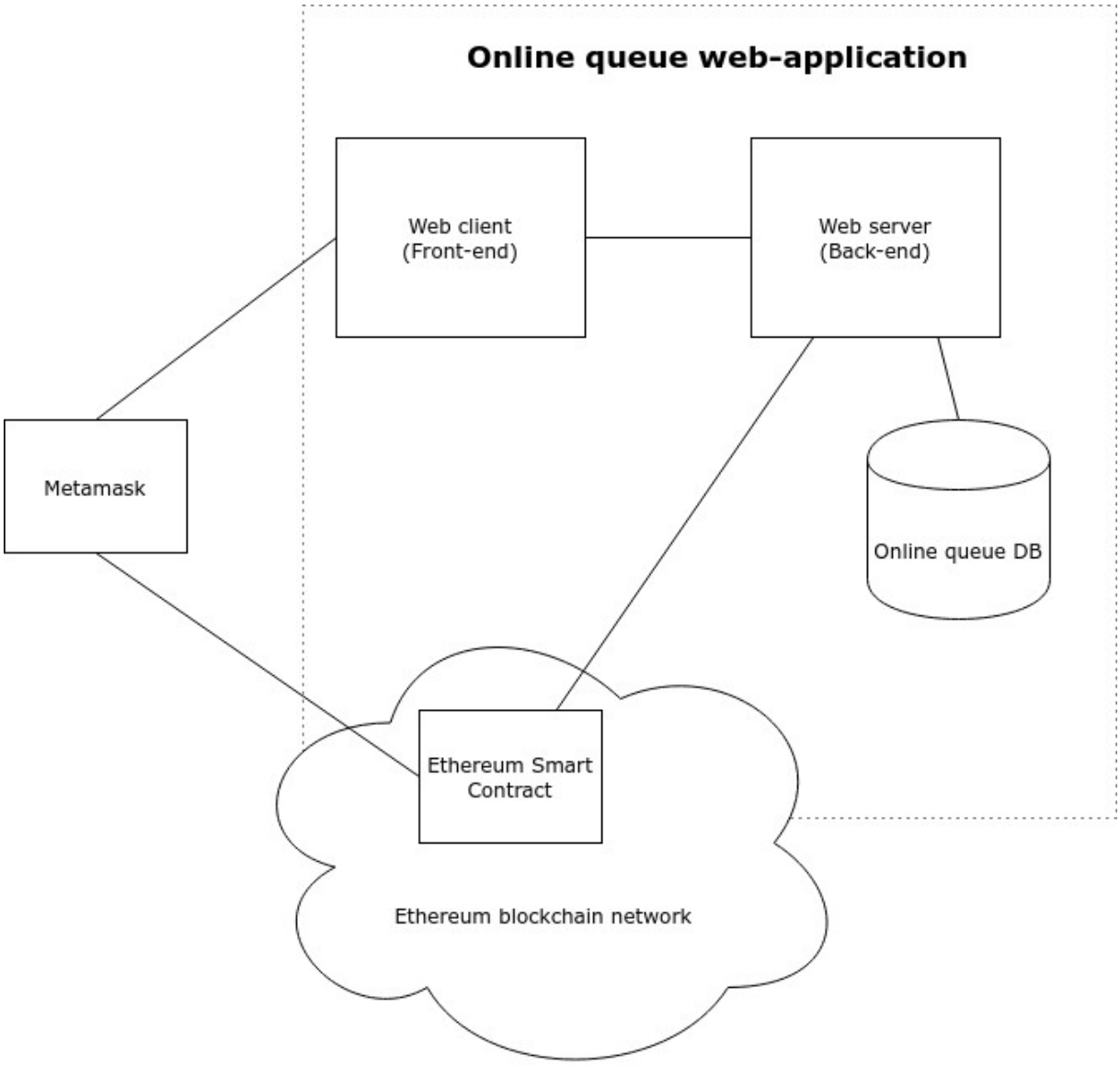


Рис. 3.1. Узагальнена структурна організація веб-додатку

Серверна частина програмного забезпечення виконує такі функції:

1. Оброблення запитів, отриманих з клієнтської частини.
2. Реалізація роботи з базою даних; видалення, збереження, оновлення даних.
3. Щоденне оновлення рейтингу спеціалістів.
4. Відправлення транзакцій до мережі Ethereum, які мають відбутися автоматично від імені сервісу.

Клієнтська частина програмного забезпечення виконує такі функції:

1. Забезпечення взаємодії з функціями системи за допомогою інтерфейса користувача.
2. Відображення даних, отриманих з сервера.
3. Візуалізація даних про рейтинг спеціалістів
4. Забезпечення запитів до блокчейну Ethereum на коректної взаємодії зі смарт-контрактами.
5. Отримання та відображення даних зі смарт-контрактів.
6. Верифікація незмінності оцінки або коментаря на основі даних із смарт-контракту за запитом користувача.

Смарт-контракти виконують наступні функції:

1. Збереження даних про запис (id запису)
2. Можливість збереження та отримання даних про оцінку візиту до спеціаліста користувачем.
3. Неможливість неавторизованого видалення даних після їх запису в смарт-контракт.

Схематично основний процес запису до онлайн-черги можна побачити на схемі, зображеній на рис. 3.2:

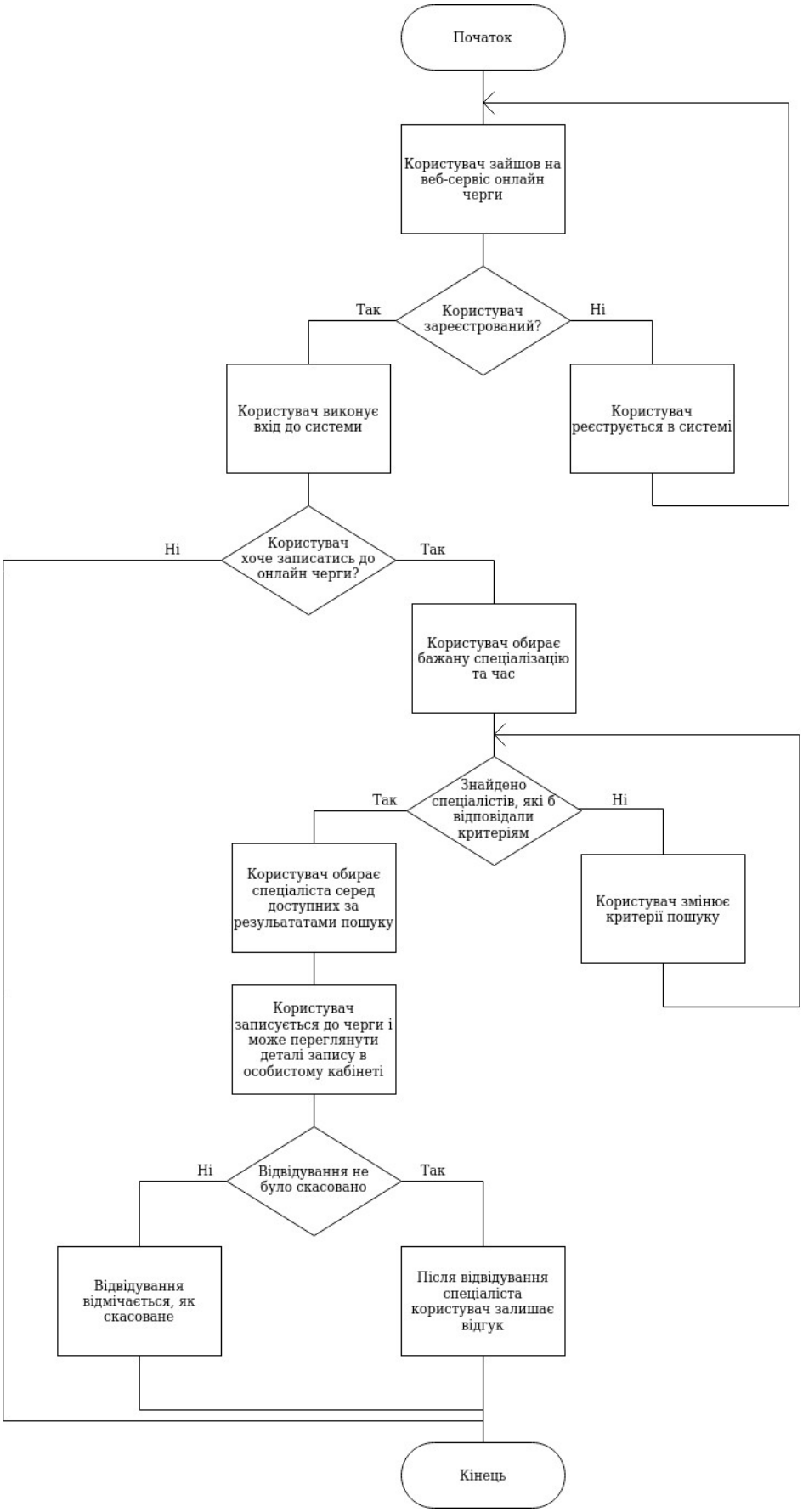


Рис. 3.2. Алгоритм запису до онлайн-черги з точки зору користувача

Основний сценарій використання системи з точки зору користувача можна описати наступним чином:

1. Користувач реєструється в системі.
2. Користувач авторизується в системі.
3. Користувач виконує пошук бажаних спеціалістів за шуканими параметрами.
4. Користувач виконує запис до обраного спеціаліста.
5. Користувач може переглянути деталі призначеного запису в своєму особистому кабінеті.
6. Користувач може скасувати запис, доки не настав час відвідування.
7. Користувач залишає оцінку та коментар, якщо відвідування відбулося.

Таким чином, веб-додаток забезпечує зберігання даних про оцінки користувачів у смарт-контракті, що дозволяє гарантувати їхню незмінність та неможливість шахрайства з рейтинговими даними.

3.2. Структура бази даних

Для організації даних обрана MongoDB. Це нереляціна БД, в якій дані збегігаються у записах, котрі організовані у вигляді документів та колекцій.

На рис. 3.3 представлена ERD-діаграма бази даних системи.

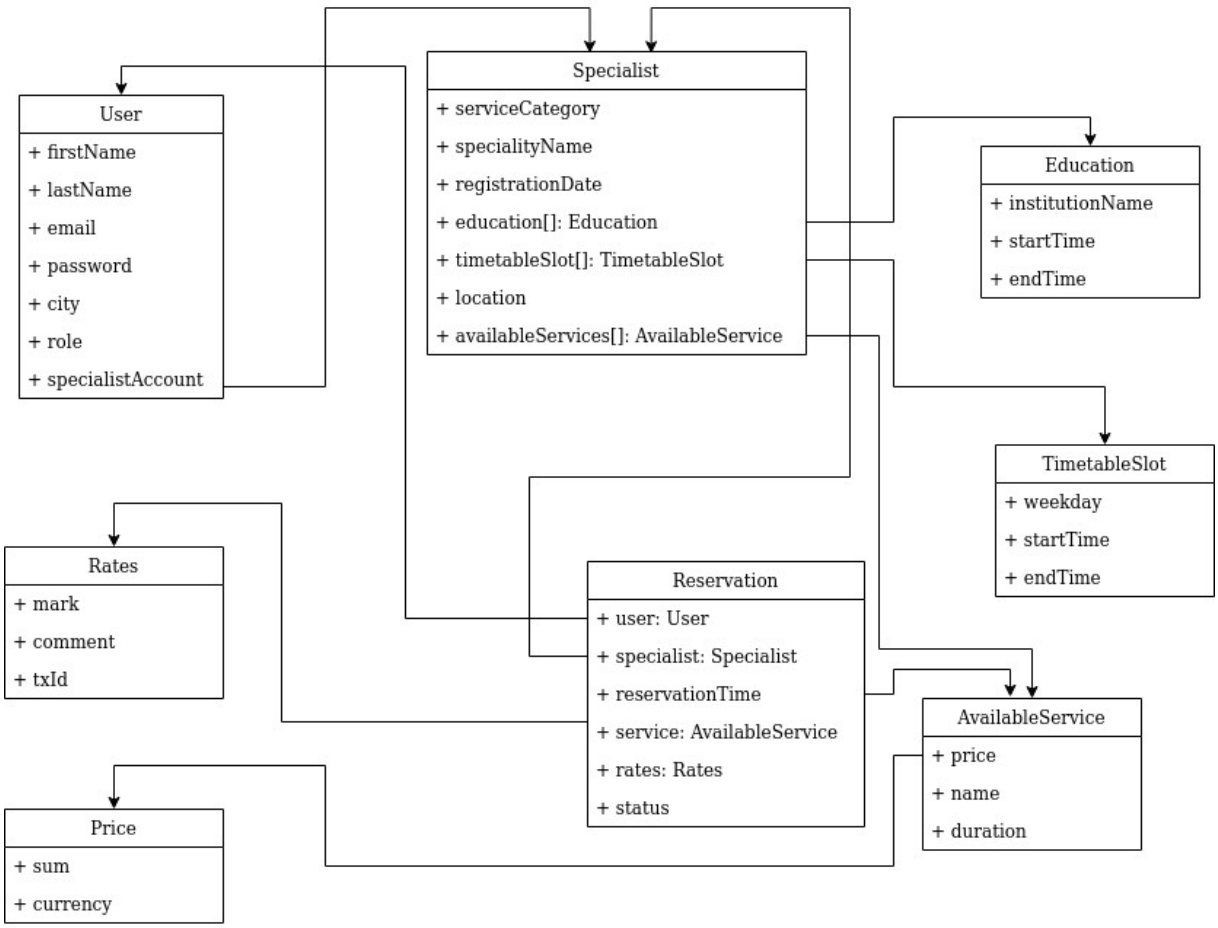


Рис. 3.3. ERD-діаграма бази даних системи

Документ User зберігає дані про користувачів:

- firstName – ім’я користувача;
- lastName – прізвище користувача;
- email – адреса електронної пошти користувача;
- password – захешований пароль для входу в систему;
- city – місто проживання користувача;
- role – роль користувача (може бути User або Specialist);
- specialistAccount – дані про користувача, як про спеціаліста (тільки, якщо користувач має роль Specialist), описуються об’єктом типу Specialist.

Документ Specialist зберігає дані про спеціалістів:

- serviceCategory – назва катерогії надання послуг, в якій працює спеціаліст (наприклад, Спорт, Краса, Фінанси та ін.);
- specialityName –назва спеціальності;
- registrationDate –дата реєстрації спеціаліста в сервісі;
- education[] – масив даних про про освіту спеціаліста, описуються записом типу Education;
- timetableSlot[] – масив даних про графік спеціаліста в межах одного дня тижня;
- location – місце роботи спеціаліста;
- availableServices[] – масив даних про послуги, які надає спеціаліст, описуються записом типу AvailableSerive.

Документ Education зберігає дані про освіту спеціаліста:

- institutionName – назва закладу освіти чи іншої установи, де проходило навчання;
- startTime – час початку навчання;
- endTime – час завершення навчання.

Документ TimetableSlot зберігає дані про графік спеціаліста в межах одного дня тижня:

- weekday – день тижня, для якого вказаний графік спеціаліста;
- startTime – час початку робочого часу;
- endTime – час завершення робого часу.

Документ AvailableSerive містить інформацію про одну з послуг, котру надає спеціаліст:

- price – вартість послуги, описується об’єктом типу Price;
- name – назва послуги;
- duration – тривалість послуги, вказується в хвилинах.

Документ Price містить інформацію про ціну послуги:

- sum – сума до сплати;
- currency – валюта, в котрій вказана ціна.

Документ Reservation містить інформацію про запис до черги:

- user – користувач, який здійснив запис на відвідування, описується об’єктом типу User;
- specialist – спеціаліст, запис до якого було здійснено, описується об’єктом типу Specialist;
- reservationTime – дата та час, на які призначено відвідування.
- serviceType – тип послуги, яку було обрано користувачем для відвідування, описується об’єктом типу AvailableSerive;
- rates – оцінка, яка була заповнена користувачем після відвідування (якщо є), описується об’єктом типу Rates;
- status – статус відвідування, може набувати значень з такого переліку CREATED, CANCELLED, WAITING_FOR_EVALUATION, PERFORMED.

Документ Rates містить дані про оцінку користувача:

- mark – числова оцінка, якою користувач оцінив відвідування, може набувати цілосисельних значень від 1 до 10 включно;
- comment – хеш коментаря, який користувач залишив про відвідування;
- txId – хеш транзакції запису даних про відгук до смарт-контракту в мережі Ethereum.

3.3. Організація взаємодії із смарт-контрактами

Розроблений та опублікований в мережі Ethereum смарт-контракт використовується для збереження даних про кожен відгук користувача. Таким чином, інформація про жоден відгук не може бути підробленою.

Для взаємодії із смарт-контрактом, користувачу необхідно встановити веб-гаманець Metamask. Він дозволятиме клієнтський частині безпечно формувати транзакції для взаємодії зі смарт-контрактом без

необхідності користувачу довіряти свій приватний ключ напряму клієнтському коду.

Для того, аби взаємодіяти із системою, користувачу потрібно буде створити або завантажити Ethereum гаманець. Це необхідно тому, що операція запису в блокчейн вимагає сплати невеликої комісії за запис даних в смарт-контракт.

При записі в чергу, в смарт-контракті створюється нова оцінка користувача. Після створення вона матиме статус незаповненої.

Після того, як призначений за розкладом сеанс було завершено, система запитує в користувача його оцінку та відгук про відвідування спеціаліста. Після того, дані відгука разом з даними про інтерфейс функції запису даних контракту, адреси контракту сереалізуються у Ethereum транзакцію, підписуються особистим ключем користувача і відправляються до мережі.

Смарт-контракт перевіряє, що відправник транзакції – дійсно той користувач, якого було вказано при створенні заявки на відгук. Чисельне значення оцінки та відгук зберігаються, статус оцінки змінюється на «заповнений». З програмної точки зору вже не можуть бути змінені чи видалені.

Якщо сеанс було скасовано чи користувач відмовився надавати оцінку, запис про сеанс буде видалено зі смарт-контракту. Подібна економія пам’яті в смарт-контракті є вигідною, оскільки комісія за транзакції залежить від кількості місця. що була використана контрактом під час виконання.

Якщо відгук та оцінка були успішно заповнені, то для цього відгуку буде наявною можливість перевірити незмінність коментаря та оцінки. При цьому, клієнтська частина виконує запит до смарт-контракту, отримує оцінку та хеш коментаря, хешує коментар, що відображається користувачу в поточний момент часу. У звіті, що буде доступний користувачеві в

діалоговому вікні, доступні будуть відповідні обчислення на посилання на транзакцію запису даних в смарт-контракт в онлайн-сервісі Etherscan.

Детальний сценарій взаємодії із смарт-контрактом наведено на рис.3.5:

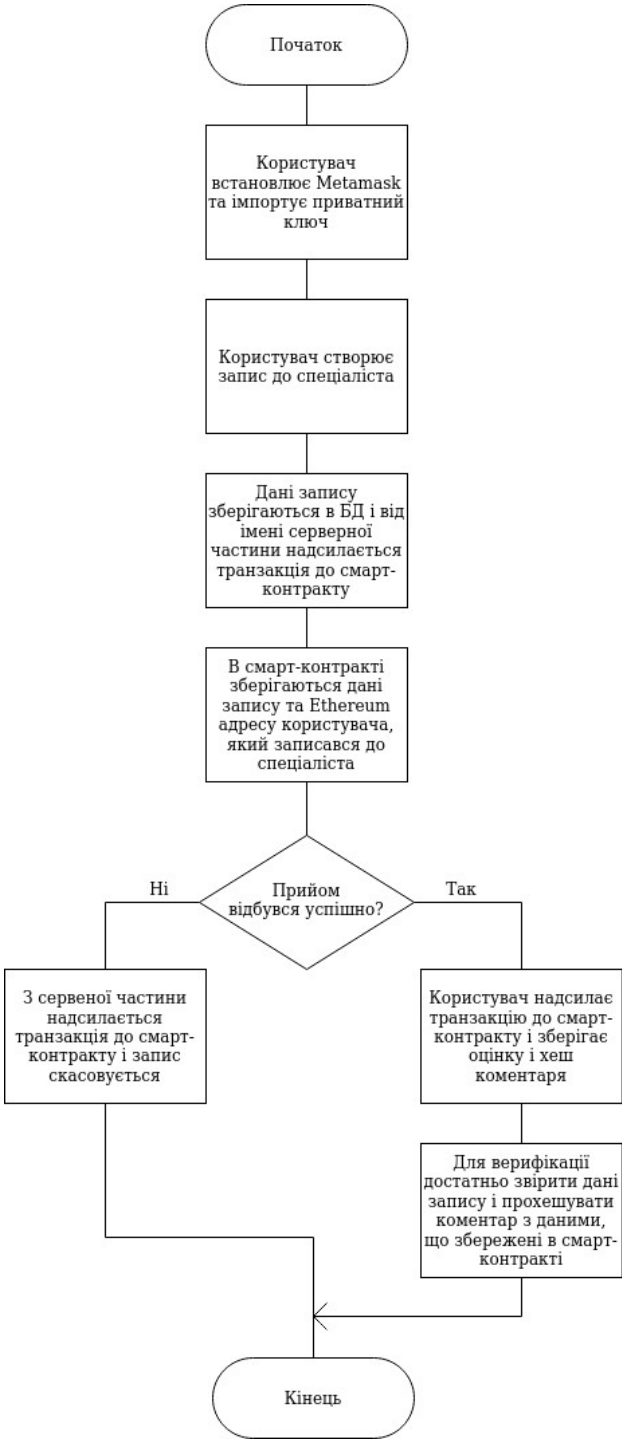


Рис. 3.5. Алгоритм взаємодії системи зі смарт-контрактами при збереженні даних про оцінку відвідування

4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Особливості реалізації серверної частини

Серверна частина розробленого додатку виконує завдання з взаємодії з базою даних, запису та збору даних з мережі Ethereum, надання доступу до даних через публічне Restful API.

Можна виділити такі модулі серверної частини:

1. API endpoints module.
2. Flask Resources classes module.
3. Ethereum interaction module.
4. Entity manipulation module.
5. Rating calcuation module.
6. MongoDB Schemas module.

Відношення між цими модулями та зовнішніми сутностями можна побачити на діаграма модулів серверної частини, представленій на рис. 4.1.

API endpoints module – модуль серверної частини, який слугує для того, аби представляти доступ до фукнціальності в форматі роутів RESTful API. У цьому модулі роутам сервера ставляться у відповідність класи ресурсів із модуля Flask Resources classes module.

Flask Resources classes module – модуль серверної частини, який слугує для зіставлення фукнціональності сервера з роутами для управління. Модуль представляє собою ряд класів, які наслідуються від класу Flask.Resource, які слугують обробниками для кожного роуту сервера, з методами для кожного HTTP методу. Кожен метод повертає стандарті HTTP відповіді з результатами виконання або даними, що були запитані.

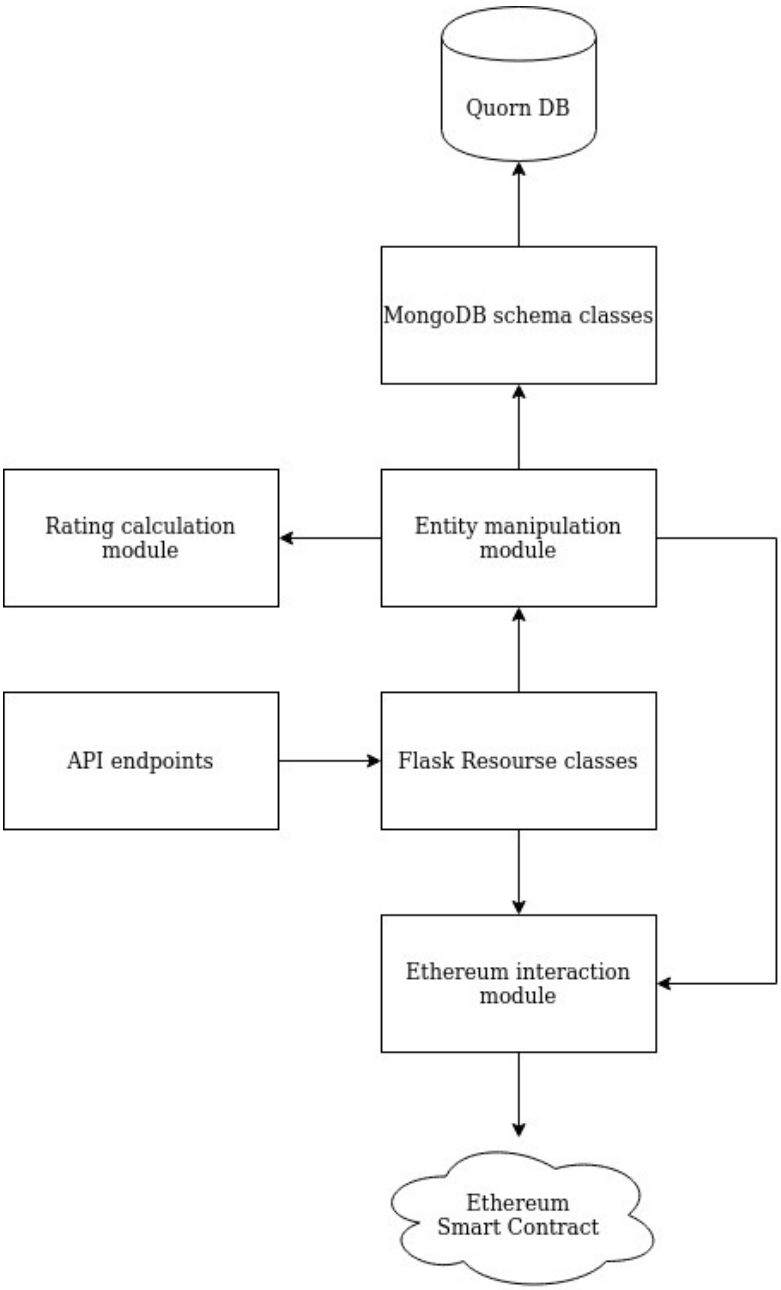


Рис. 4.1. Діаграма модулів серверної частини

Ethereum interaction module – модуль серверної частини, який відповідає за взаємодію із блокчейн-мережею Ethereum. Взаємодія організовується за допомогою бібліотеки web3.py. Транзакції надсилаються від імені аккаунта системи, приватний ключ якого зберігається, як змінна оточення серверу задля безпеки даних.

Entity manipulation module – модуль серверної частини, який відповідає за оформлення сирих даних, що отримуються з бази даних.

Даний модуль складається з підмодулів, які відповідають за сегменти даних системи: User, Specialist, Timeslots, Rates, Reservations. Також, він реалізує додаткову логічну фукнціональність, якщо така необхідна, наприклад, метод для аутентифікації користувача і генерування jwt токену.

Rating calcuation module – модуль серверної частини, який відповідає за підрахунок рейтингу спеціаліста.

MongoDB Schemas module – модуль серверної частини, який описує схеми документів БД Mongo, створює зв'язок з віддаленою БД, викорує CRUD операції. Для опису схем документів та взаємодія з БД відбувається за допомогою бібліотеки mongoengine.

4.2. Особливості реалізації клієнтської частини

Клієнська частина розробленого додатку виконує завдання з представлення інформації користувачеві на веб-сторінці та передачі його вводу серверній частині та смарт-контракту в мережі Ethereum.

Можна виділити такі модулі клієнтської частини:

1. Layout module.
2. Routing module.
3. Providers module.
4. Pages module.
5. Constants module.
6. Hooks module.
7. Redux store module.
8. API interactions module.

Відношення між цими модулями та зовнішніми сутностями можна побачити на діаграма модулів клієнської частини, представленій на рис.4.2.

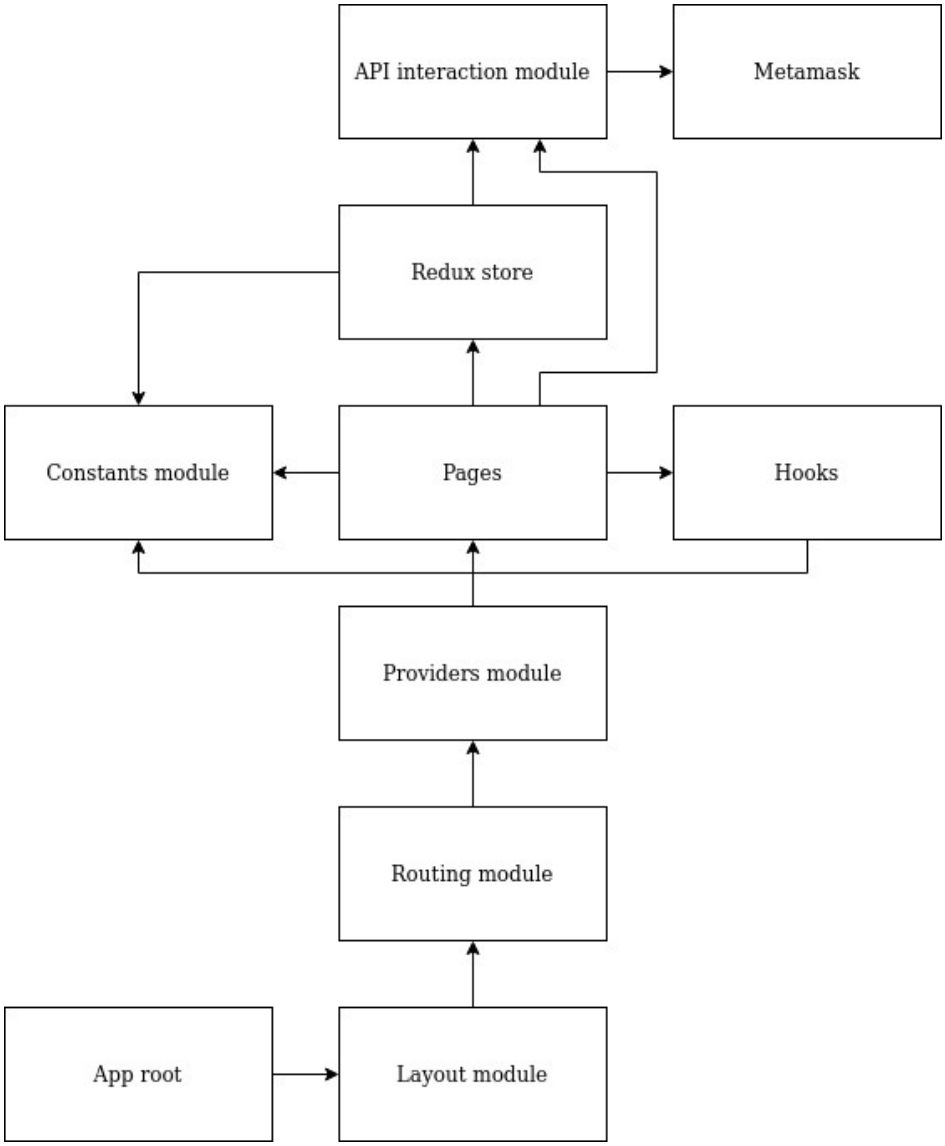


Рис. 4.2. Діаграма модулів клієнтської частини

Layout module – модуль клієнтської частини, який відповідає за розташування основних елементів веб-сторінки, яких як хедер, футер, бокове меню, основна частина частина сторінки, тощо.

Routing module – модуль клієнтської частини, що відповідає за демонстрування основної частини сторінки в залежності від обраної адреси, а також оброблює ситуації, коли потрібна сторінка не знайдена, користувач не авторизований, тощо.

Providers module – модуль клієнтської частини, який декорує деякі сторінки додатковими перевірками, наприклад, виконує показ сторінки,

якщо користувач не авторизований і перенаправляє на сторінку авторизації, якщо не авторизований.

Pages module – модуль клієнтської частини, що містить безспосередньо основні сторінки сервісу. Сторінки відображають дані та отимують ввід користувача, а для виконання операцій отримання та відправки даних, запити на ввід передаються до Redux store module.

Constants module – модуль клієнтської частини, що містить константи, необхідні для роботи системи, наприклад посилання на адресу API серверу.

Hooks module – модуль клієнтської частини, в якій винесені React Hooks, що використовуються в компонентах-сторінках.

Redux store module – модуль клієнтської частини, який відповідає за збереження стану додатку та зберігання глобальних даних і їхнє оновлення. Для цього використовується фреймворк React redux. У модулі описані state, redusers та actions, що компонуються разом і прив’язуються до усіх компонент, де можуть знадобитися певні дані або виконання певних дій.

API interactions module – модуль клієнтської частини, який відповідає за взаємодію із мережею. Через цей модуль надсилаються запити до API серверної частини та запити до розширення Metamask на надсилання транзакцій до мережі Ethereum.

4.3. Особливості реалізації смарт-контрактів

Смарт-контракт зберігає дані про оцінку на відгук щодо кожного відвідування, що успішно відбулося. Ці дані зберігаються у структурі даних mapping, одній зі стандартних структур даних мови Solidity, яка у відповідність ідентифікатору запису ставить структуру Rate.

Структура Rate містить поля:

- status – статус відгуку з тих, що визначені в переліку Status;

- mark – числова оцінка користувача;
- commentHash – хеш коментаря користувача;
- timestamp – час заповнення відгуку;
- visitorAddress – Ethereum адрес відвідувача;
- specialistStamp – хеш від деяких особистих даних спеціаліста, що служить його ідентифікатором в системі.

Перелік Status, який визначає можливі статуси відгуку може набувати таких значень:

- 0, UNSET – відвідування с таким ідентифікатором не було ініціалізовано або ж скасоване;
- 1, PENDING – відвідування було створене і воно очікує запису відгуку;
- 2, COMPLETED – відгук про відвідування було успішно записано.

Смарт-контракт має такі функції:

- rate() – отримати дані про відгук за ідентифікатором;
- registerAppointment() – зареєструвати запис в смарт-контракті. Важливим є те, що переданий при створенні запису Ethereum адреса користувача обов’язково має бути коректною, оскільки тільки з цієї адреси можна бути записати дані про відгук. Функція може бути викликана лише з адреси, яка має права адміністратора контракту;
- setAppointmentRates() – записати відруг у сматрт-контракт. Функція може бути викликана лише з адреси, що вказана при створенні запису, як адреса відвідувача;
- cancelAppointment() – видалити запис із смарт-контракту. Видалити запис можна лише тоді, якщо він має статус PENDING. Функція може бути викликана лише з адреси, яка має права адміністратора контракту.

Смарт-контракт реалізує шаблон проектування контрактів Ownable та Adminable. Це означає, що при розгортанні контракту адреса надсилача транзакції запам’ятовується як адреса власника контракту. В подальшому тільки власник контракту може викликати певні функції, що вимагають привілейованого доступу, наприклад, додавати адміністраторів контракту.

Структуру наслідування контрактів можна побачити на рис.4.3.

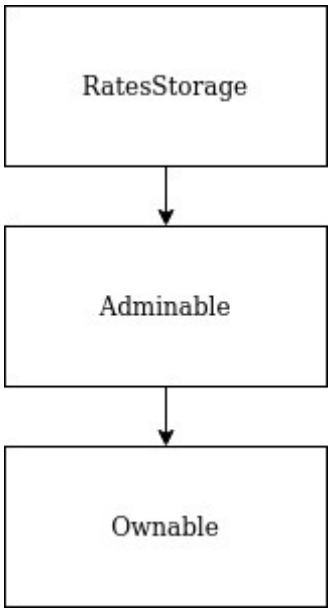


Рис.4.3. Структура наслідування смарт-контрактів

Смарт-контакт було успішно опубліковано в одній з тестових мереж Ethereum, Ropsten Network за адресою:

0x6ad25cB063bC6EBBC7A0eD66CBB91AA4C7FAd86E.

Будь-хто може переглянути статус контракту, використовуючи публічно доступний сервіс Etherscan [18].

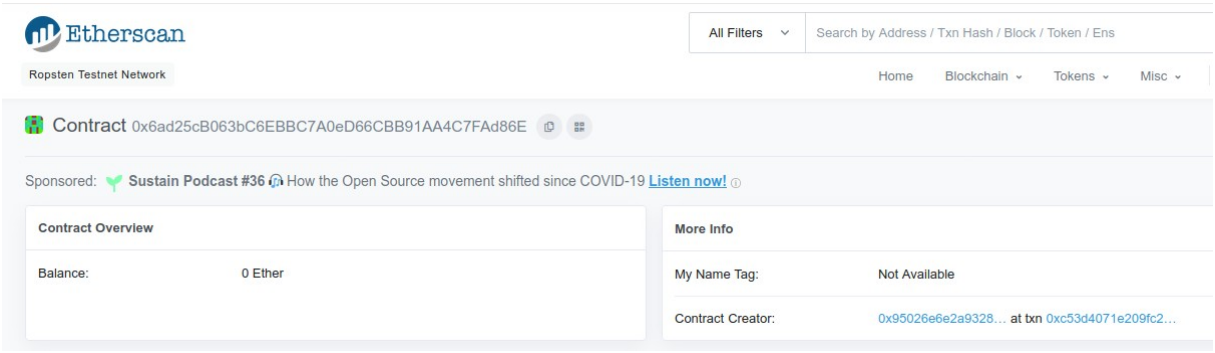


Рис. 4.4. Сторінка смарт-контракту на Etherscan

Варто зазначити, що можливість видалити контракт з мережі не передбачена програмно, а отже не є можливою. Те ж саме стосується і усіх відгуків, що там було успішно збережено. З огляду на це, цей контракт та усі збережені у ньому відгуки існуватимуть в мережі Ethereum доти, доки існуватиме мережа.

4.4. Дизайн та вміст веб-сторінок

При потраплянні на веб-сервіс неавторизований користувач буде перенаправлений на сторінку логіну. Зовнішній вигляд сторінки логіну зображено на рис 4.5.

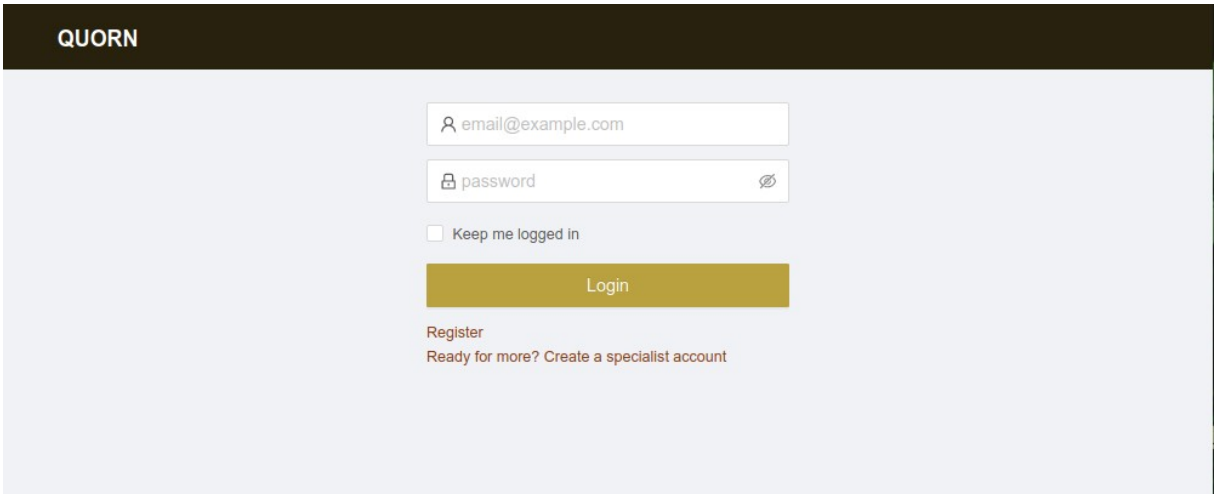


Рис. 4.5. Сторінка логіну

Якщо користувач вводить коректні електнонну адресу та пароль, він отримує повідомлення про те, що авторизація відбулась успішно.

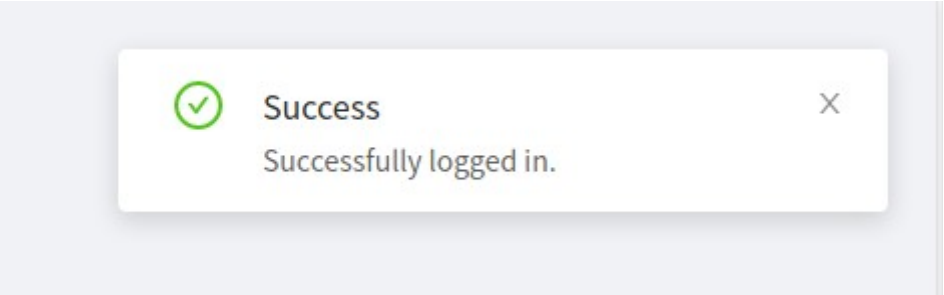


Рис. 4.6. Повідомлення про успішну авторизацію

Якщо користувач вводить неправильний пароль або під час аутентифікації стається інша помилка, він отримує про це повідомлення:

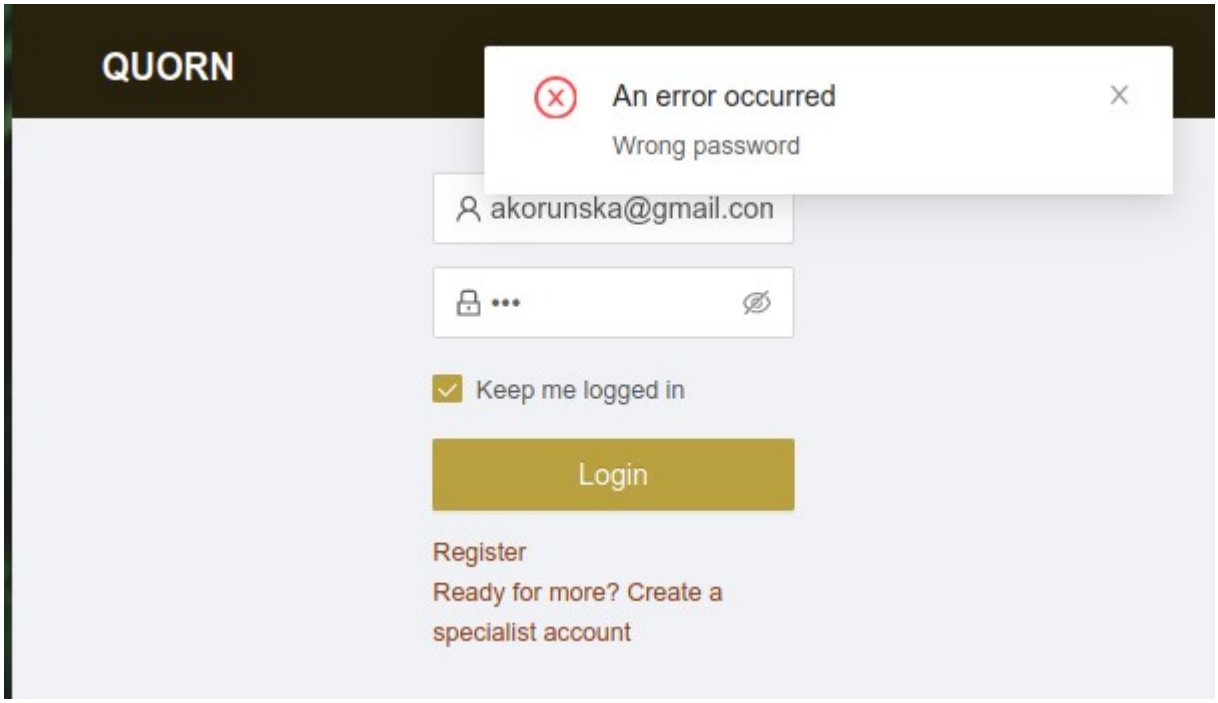


Рис. 4.7. Повідомлення про невірно введений пароль

Зі сторінки логіну незареєстрований користувач може перейти на сторінку реєстрації та зареєструватися як користувач або ж як спеціаліст.

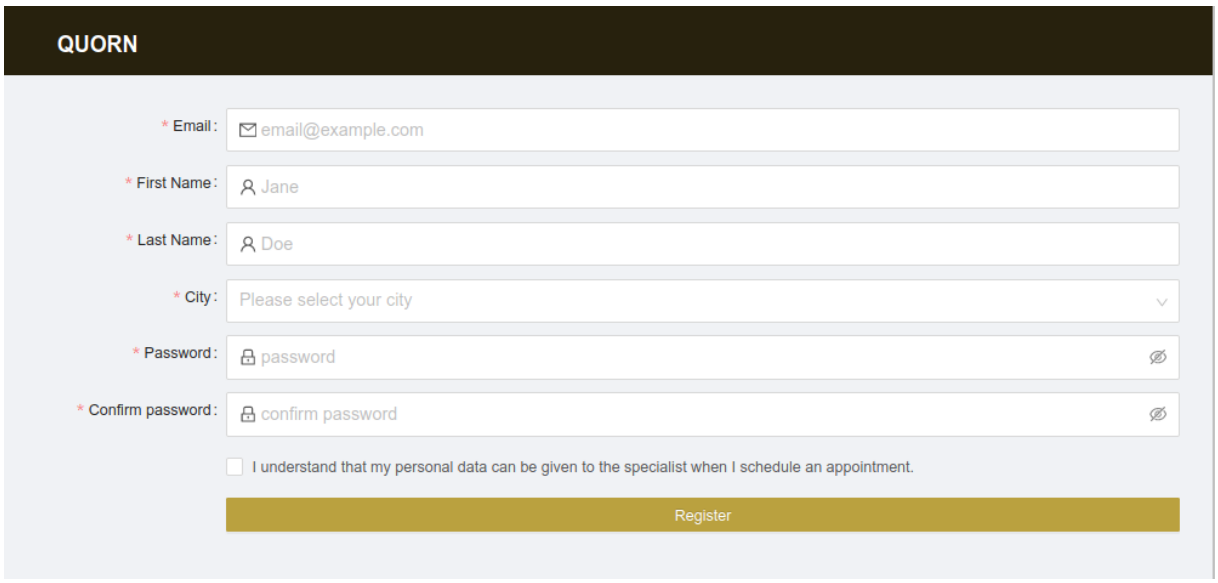


Рис. 4.8. Сторінка реєстрації користувача

Після успішної авторизації користувач перенаправляється на сторінку пошуку, що зображена на рис. 4.9. Ця сторінка дозволяє шутати спеціалістів за різними критеріями, на ній наявні

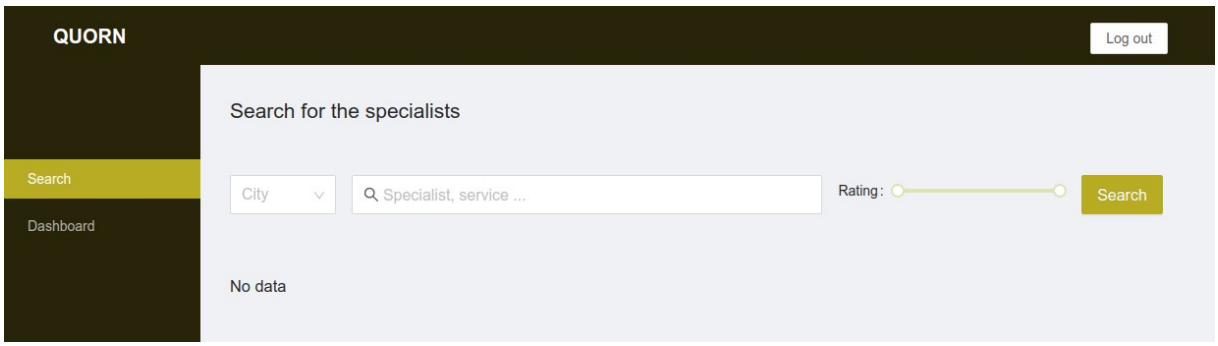


Рис. 4.9 Сторінка пошуку спеціалістів

Під час завантаження результатів пошуку відображається динамічне зображення, яке можна побачити на рис. 4.10.

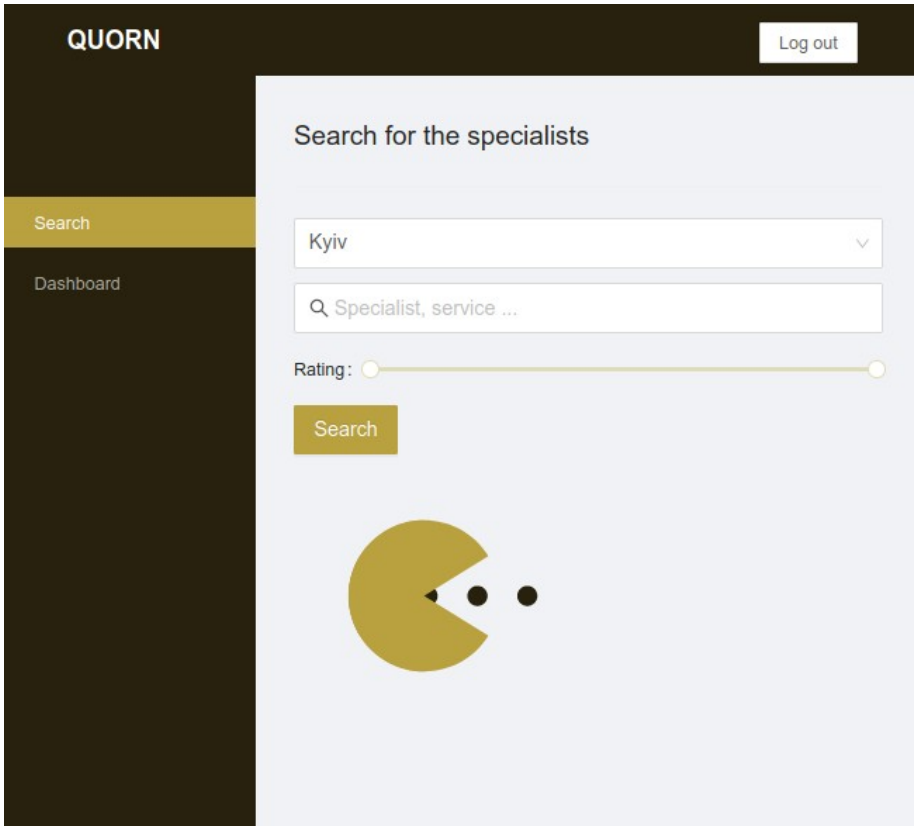


Рис. 4.10 Динамічне зображення при завантаженні результатів пошуку

В результатах пошуку відображається найголовніша інформація про спеціалістів: сфера роботи, спеціалізація, ім’я та прізвище, рейтинг та перелік послуг.

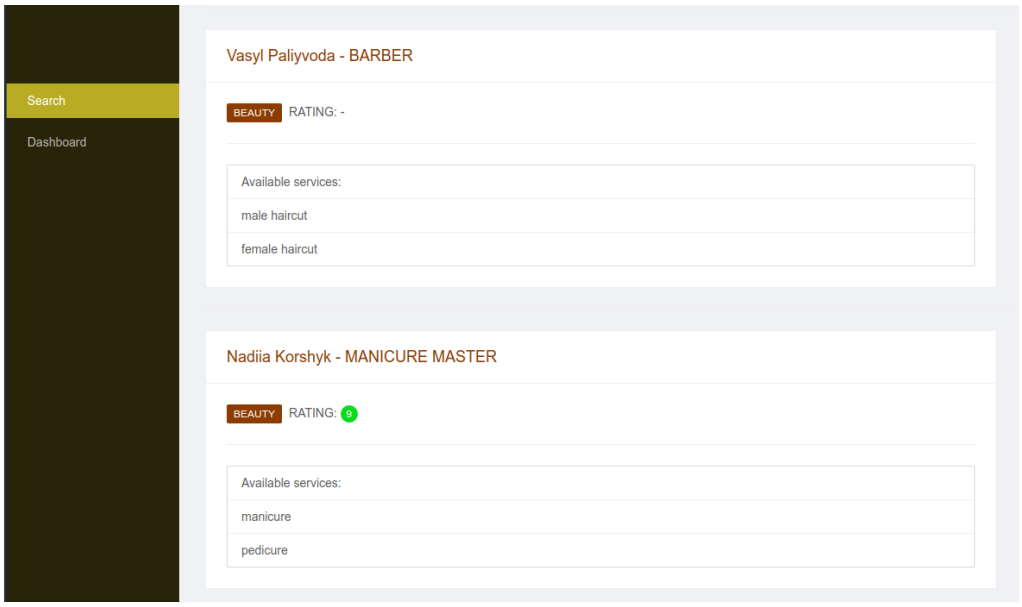


Рис. 4.11 Результати пошуку спеціаліста

При переході за посиланням із результатів пошуку, можна потрапити на сторінку спеціаліста.

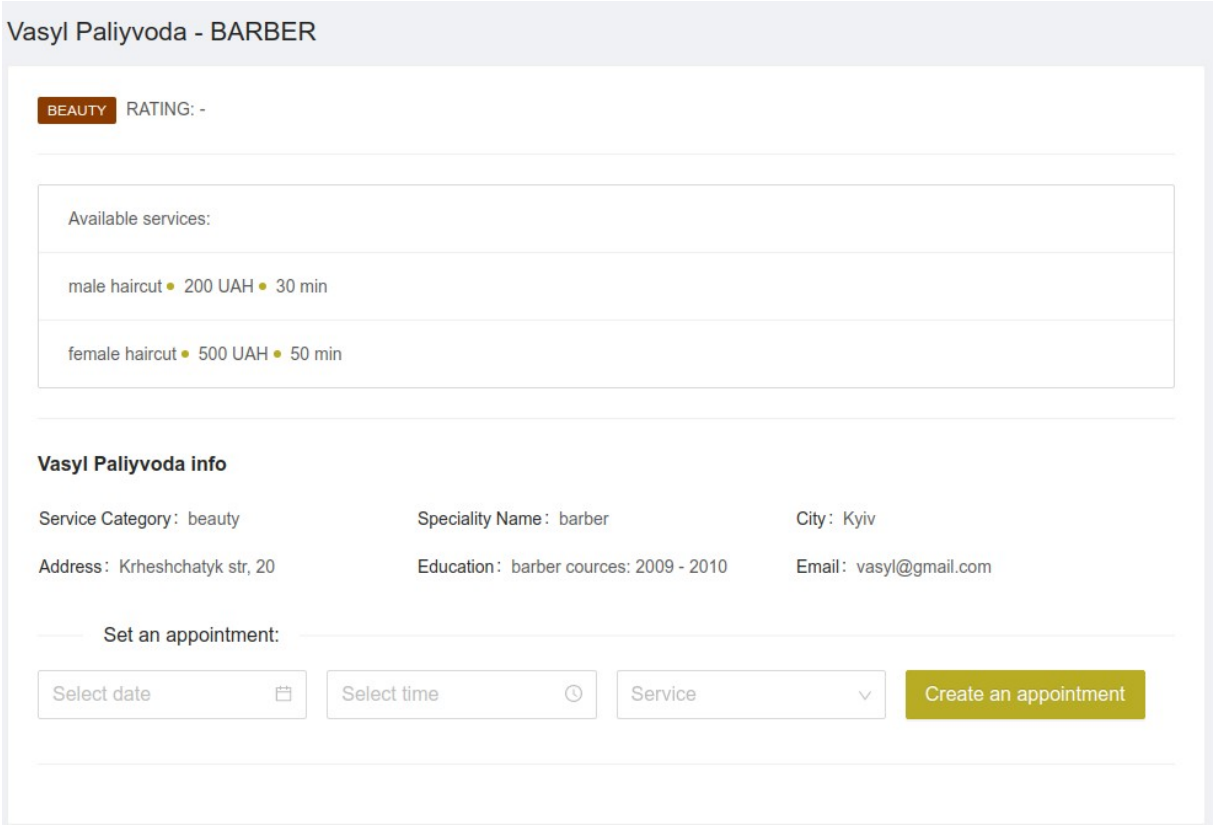


Рис. 4.12 Сторінка спеціаліста

Якщо користувач створить запис у черзі, інформація про запис з’явиться і в особистому кабінеті користувача (рис.4.13), і в особистому кабінеті спеціаліста (рис. 4.15).

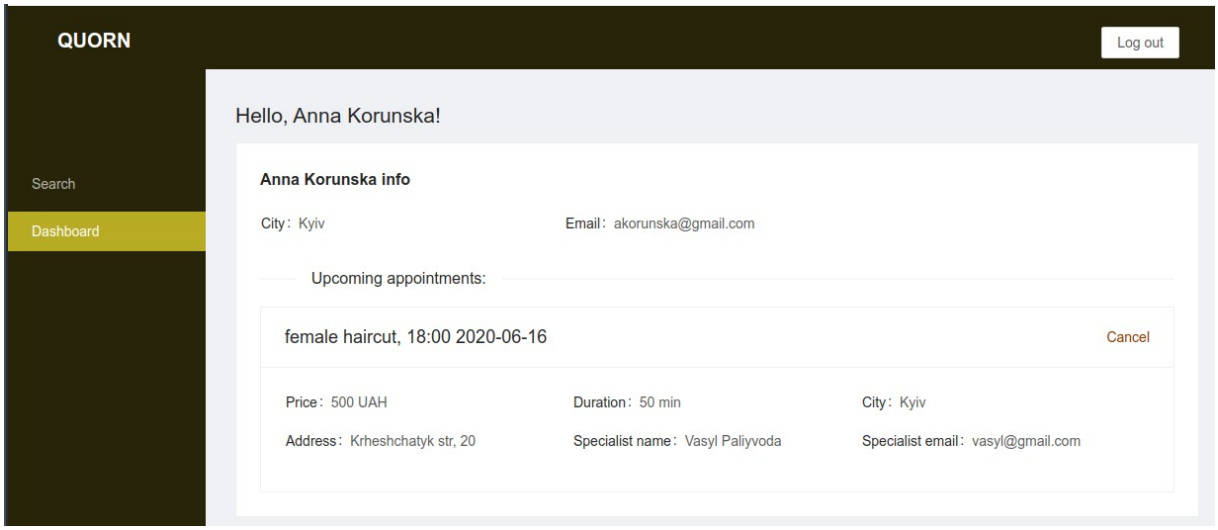


Рис. 4.13 Особистий кабінет користувача з інформацією про заплановані записи

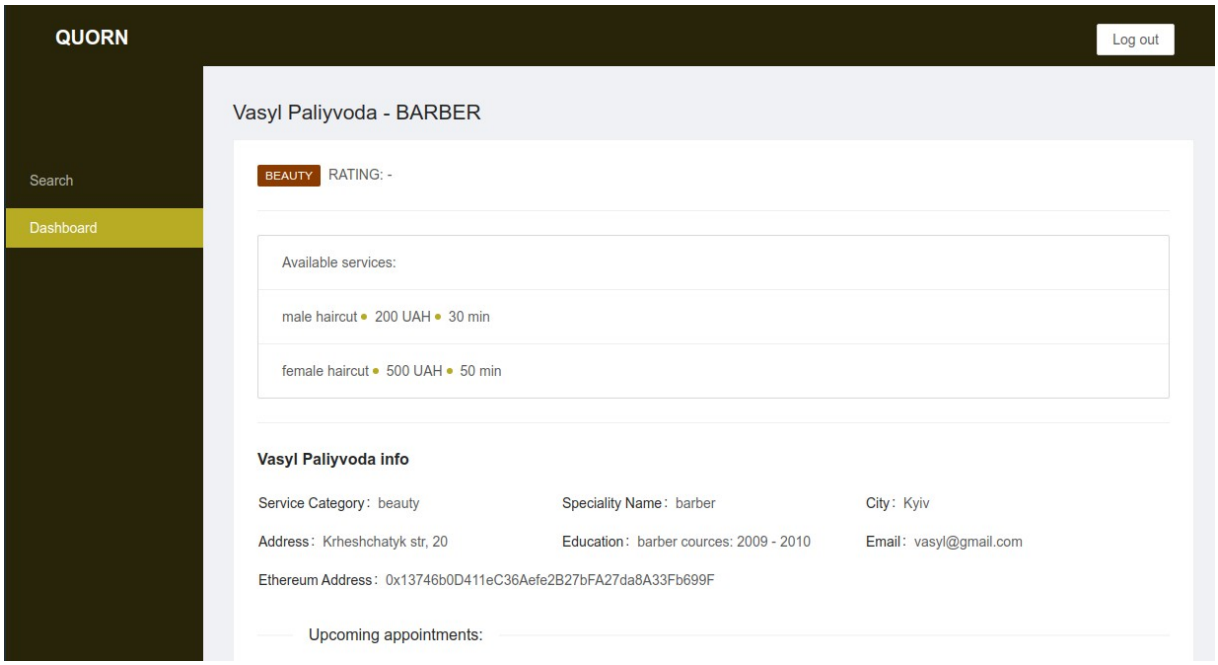


Рис. 4.14 Особистий кабінет спеціаліста

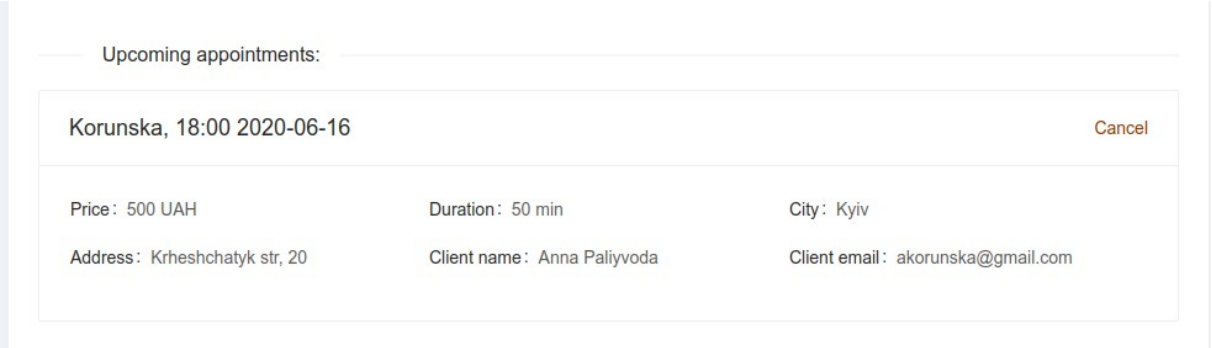


Рис. 4.15 Інформація про заплановані записи в особистому кабінет спеціаліста

4.5. Рекомендації щодо подальшого вдосконалення

Після аналізу розробленої системи можна виділити декілька шляхів подальшого вдосконалення проєкту.

Однією ж важливих фукнціональних можливостей, яка б покращила користування розробленим додатком, можна назвати оплату обраних послуг через веб-сервіс. Таким чином, оплата зписувалась би з користувача при реєстрації на відвідування і знаходилася б на проміжному рахунку, доки відвідування не відбудеться успішно, після чого кошти б перенаправлялися на рахунок спеціаліста.

Крім того, в поточній версії ПЗ допускається, що користувачі самостійно сплачують вартість gas при запису інформації в блокчейн Ethereum. Для підвищення привабливості системи для потенційних клієнтів та зменшення порогу входження для повноцінного користування системою варто розробити систему компенсування газу за рахунок системи.

Також, забезпечення роботи з більшою кількістю блокчейн-мереж (наприклад, Ontology або EOS) може бути одним із шляхів подальшого розвитку системи.

Розроблення Telegram-боту для більшої зручності користування також є пріоритетним напрямом подальшої розробки. Він міг би

забезпечувати зручний спосіб перевіряти та змінювати записи у чергу, а також нотифікувати користувачів про їхні заплановані записи.

ВИСНОВКИ

Метою даного дипломного проєкту є підвищення довіри користувачів онлайн-черг до відгуків та оцінок про роботу закладів чи окремих спеціалістів шляхом розроблення нового веб-сервісу, що гарантує неможливість шахрайства з даними чи їх цензурування.

Під час роботи над проектом було проаналізовано існуючі аналоги програмної системи та виявлено, що веб-додаток має аналоги, сильно орієнтовані на сферу охорони здоров’я і не реалізують складних підходів протидії шахрайству.

Після аналізу інструментів розроблення було обрано мову програмування Python та фреймворк Python Flask для серверної частини та бібліотеку React.js для клієнтської частини. Також, було обгрунтовано доцільність використання технології блокчейн для досягнення поставленої мети протидії шахрайству. Для інтеграції з веб-додатком було обрано мережу Ethereum із смарт-контрактами мовою Solidity.

Було розроблено та наведено структуру веб-додатку, діаграми модулів кожного компонента системи, а також діаграму бази даних.

Розроблений веб-додаток онлайн-черги забезпечує зберігання даних про реальний рейтинг закладів чи окремих спеціалістів, захищає їх від небажаного редагування після публікації, від видалення низьких оцінок або іншого роду спотворення, а отже, сервіс може гарантувати надійність і об'єктивність рейтингових даних. Прозорість побудови рейтингу та неможливість підробки коментарів є можливою завдяки використанню смарт-контрактів. Така властивість є дуже важливою для користувачів онлайн-черги, а також стимулює самих спеціалістів краще працювати і не розраховувати на нечесне підвищення свого рейтингу. Отже, можна сказати, що мету протидії шахрайству та цензуруванню було досягнуто.

Також, було наведено приклади зовнішнього вигляду інтерфеса користувача, описано процес тестування додатку та смарт-контрактів, зазначено подальші шляхи вдосконалення розробленої системи. Вимоги, які було сформульовано перед розробленням, виконані в повній мірі і в результаті тестування не було виявлено проблем.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

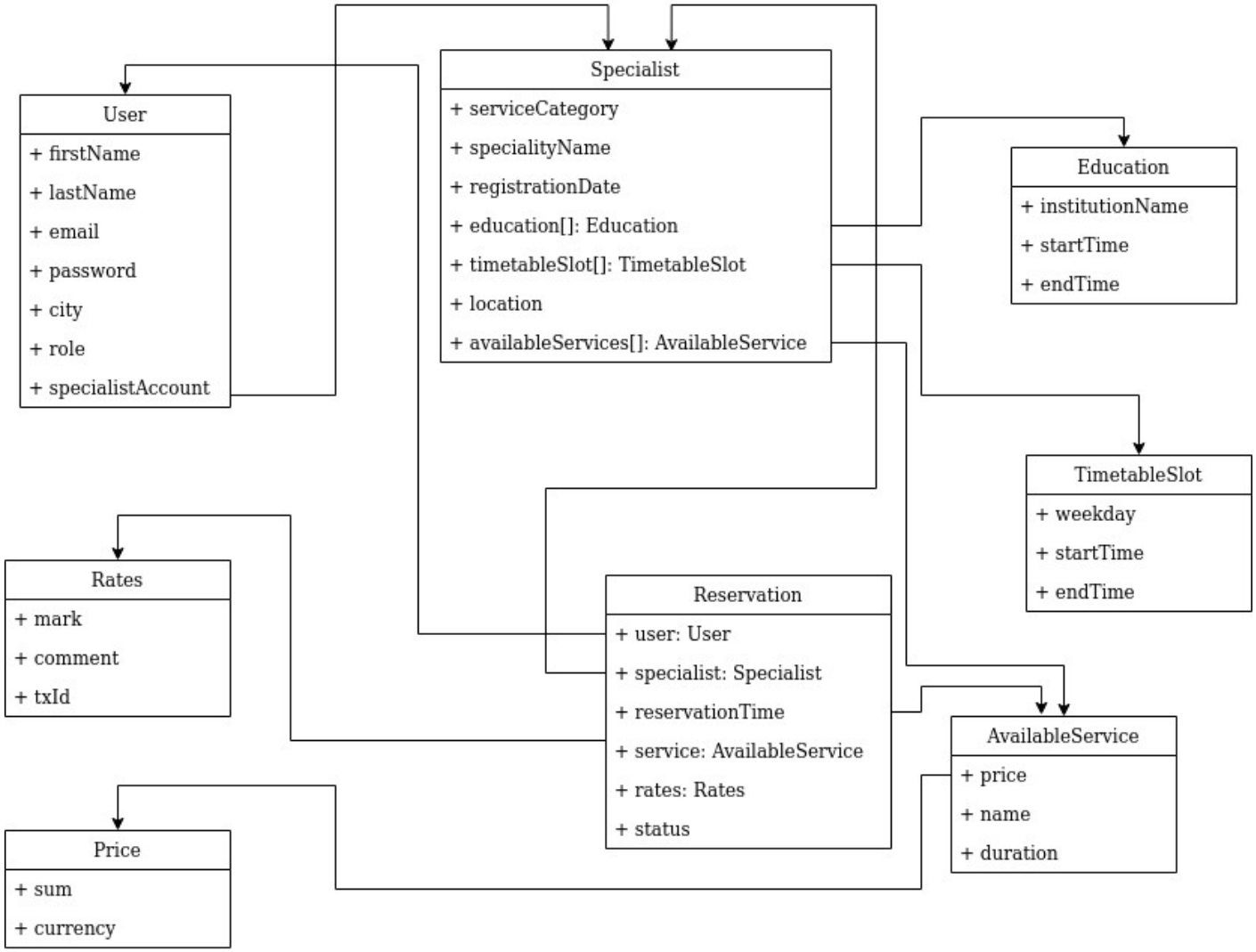
1. Електронна черга [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Електронна_черга
2. Doc.ua [Електронний ресурс]. – Режим доступу: https://doc.ua/
3. Python 3.8.3 documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/
4. Python Flask documentation [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/en/1.1.x/
5. API Reference Documentation Node.js [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs/
6. Express.js Documentatation [Електронний ресурс]. – Режим доступу: https://expressjs.com/
7. React.js documentation [Електронний ресурс]. – Режим доступу: https://reactjs.org/
8. AngularJS API Docs [Електронний ресурс]. – Режим доступу: https://docs.angularjs.org/api
9. Кравченко П. Блокчейн і децентралізовані системи : навч. посібник у 3 ч. Ч. 1 / П. Кравченко, Б. Скрябін, О. Дубініна. – Харків : ПРОМАРТ, 2019. – 452 с.
10. Andreas M. Antonopoulos. 2014. Mastering Bitcoin: Unlocking Digital Crypto-Currencies (1st. ed.). O'Reilly Media, Inc.
11. Ethereum [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Ethereum
12. CoinMarcetCap Ethereum Capitalisation [Електронний ресурс]. – Режим доступу: https://coinmarketcap.com/currencies/ethereum/
13. Solidity Ethereum documentation [Електронний ресурс]. https://solidity.readthedocs.io/en/v0.6.9/
14. OpenZeppelin Contracts [Електронний ресурс]. – Режим доступу: https://github.com/OpenZeppelin/openzeppelin-contracts

15. Remix Ethererum IDE [Електронний ресурс]. – Режим доступу: https://remix.ethereum.org/
16. Truffle documentation [Електронний ресурс]. – Режим доступу: https://www.trufflesuite.com/docs/truffle/overview
17. Ganache documentation [Електронний ресурс]. – Режим доступу: https://www.trufflesuite.com/ganache
18. Etherscan Block Explorer [Електронний ресурс]. – Режим доступу: https://etherscan.io/

ДОДАТКИ

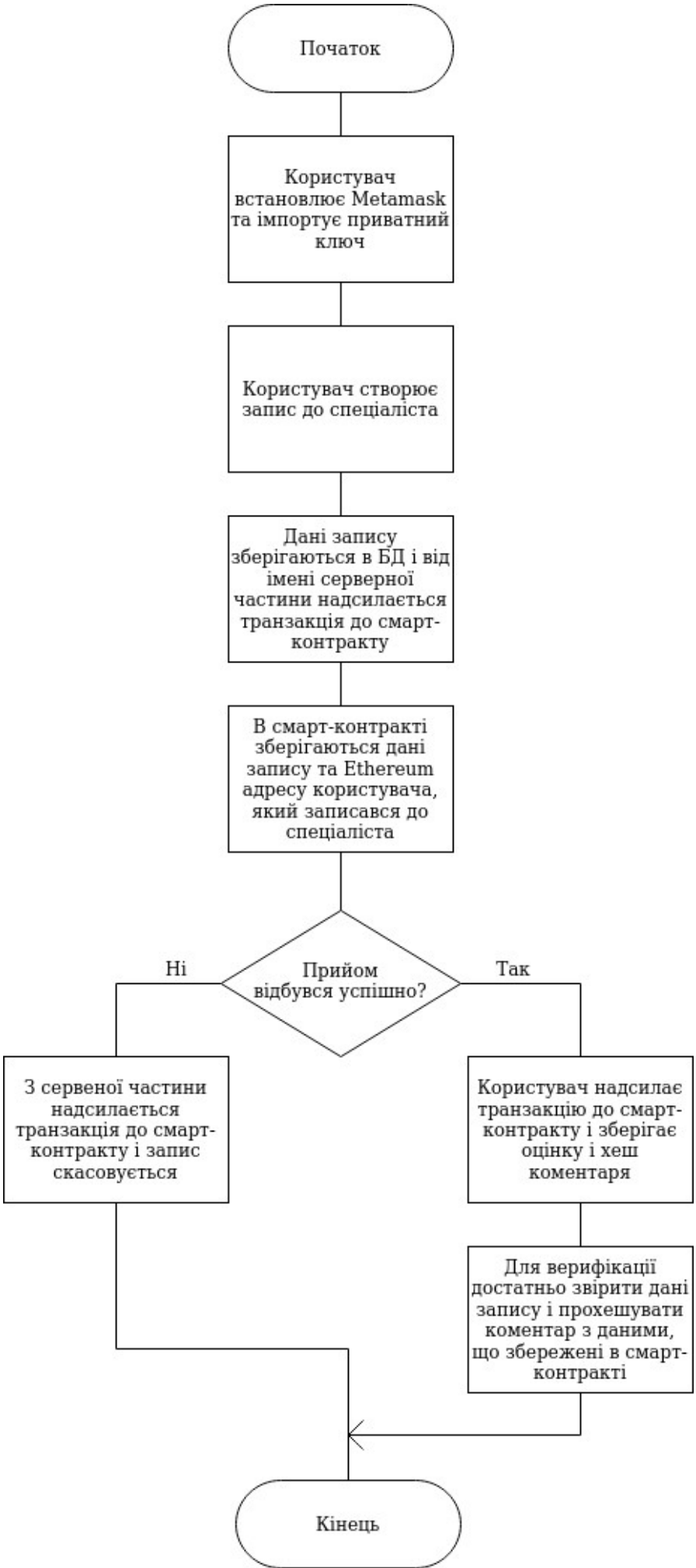
Додаток 1

Копії графічних матеріалів



ДП.045440-06-99

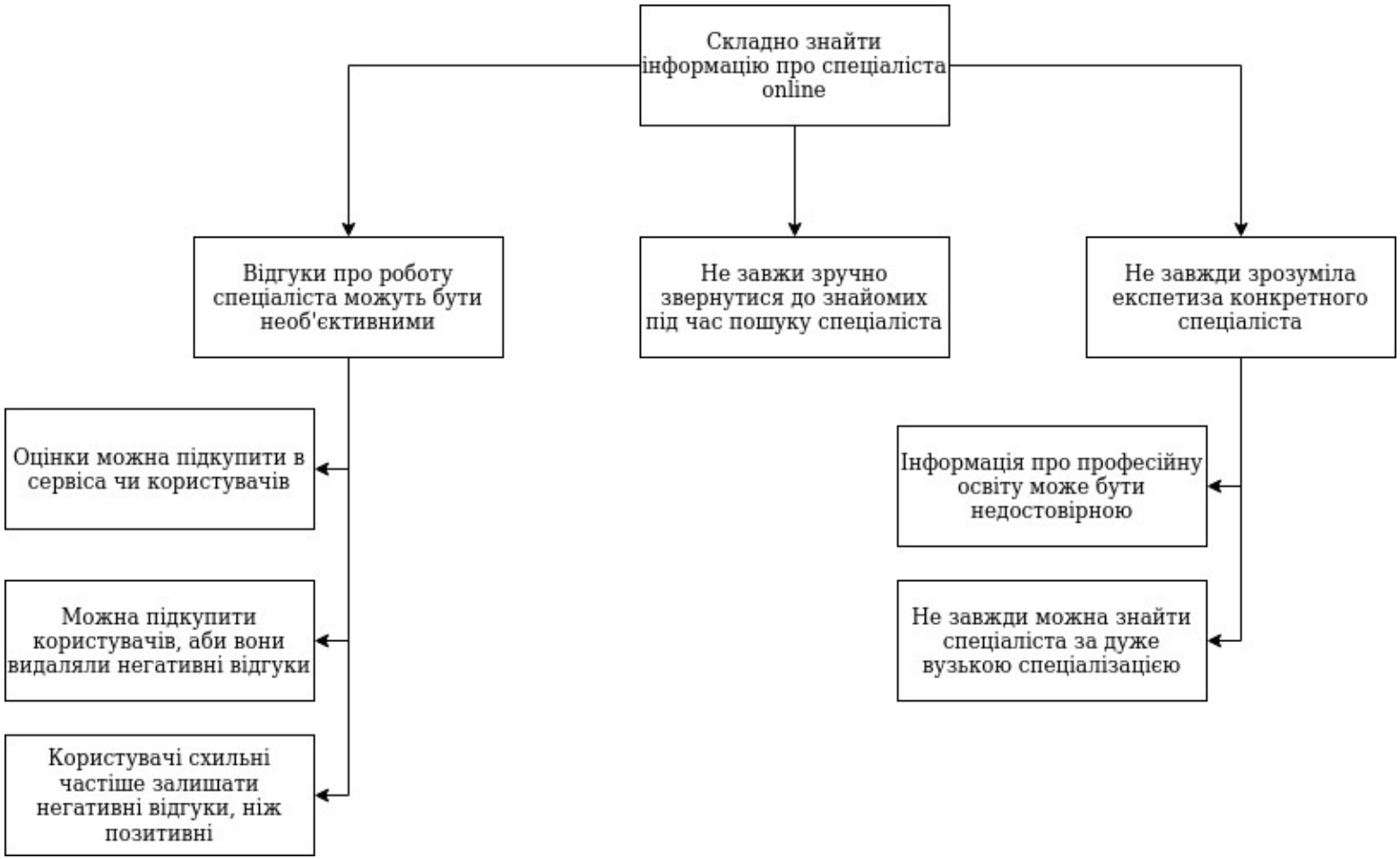
Веб-додаток для організації онлайн-черги з підтримкою можливості оцінювання якості послуг. Структура бази даних. ERD-діаграма



ДП.045440-07-99

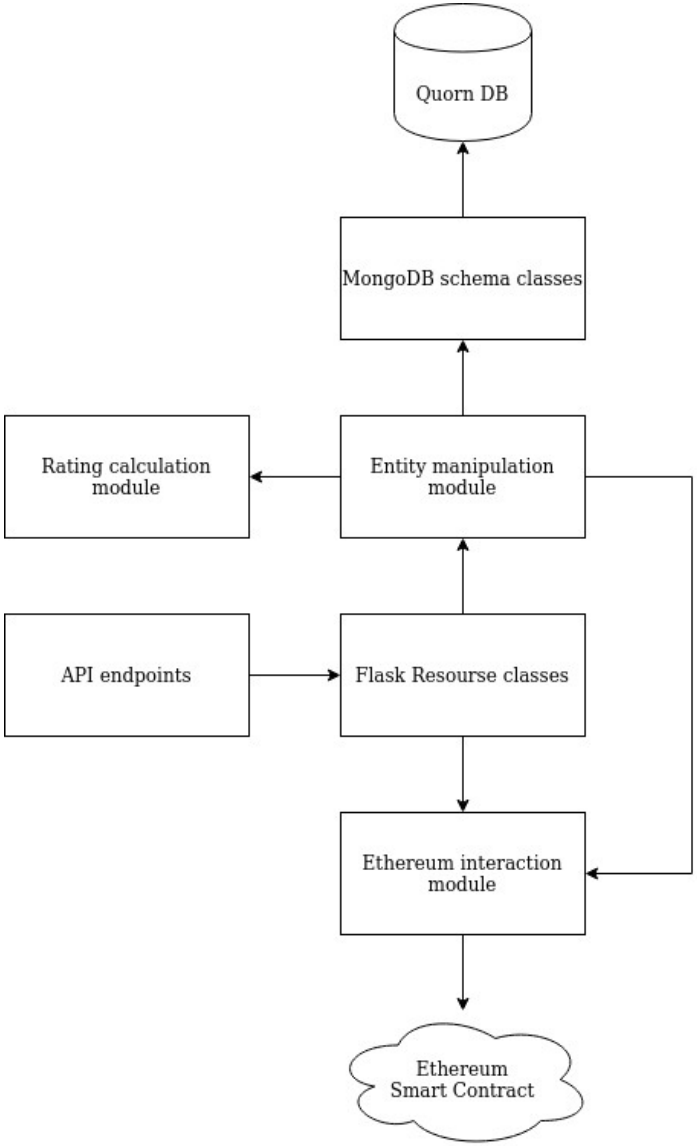
Веб-додаток для організації онлайн-черги з підтримкою можливості оцінювання якості послуг. Взаємодія системи зі смарт-контрактом. Блок-схема алгоритму

Дерево проблем

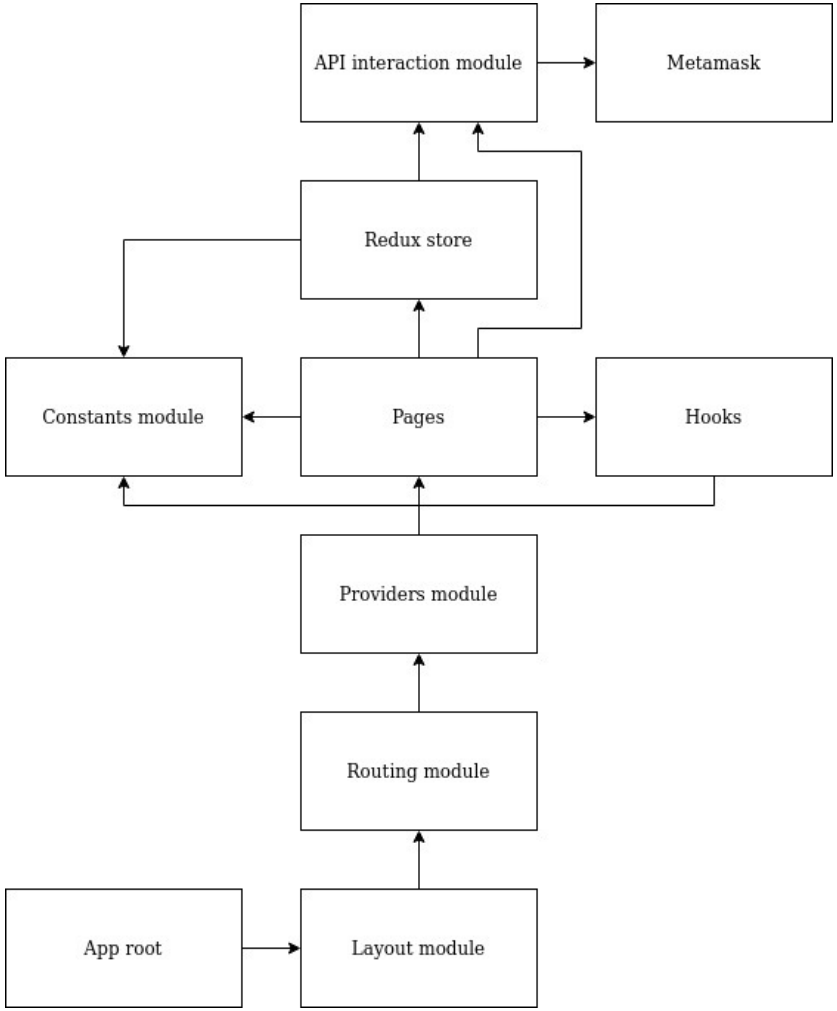


Корунська Анна, група КП-61

Структура модулів серверної частини



Структура модулів клієнтської частини



Корунська Анна, група КП-61

Додаток 2

Лістинг програми

Фрагменти коду клієнтської частини

```
import React from 'react';
import { Route, Switch } from 'react-router-dom';
import Loadable from 'react-loadable';
import AppLayout from './components/layout/AppLayout';
import { Loader } from './components/Loader';
import Authorization from './providers/Authorization';
import 'antd/dist/antd.less';
import 'ant-design-pro/dist/ant-design-pro.css';
import './assets/styles/index.scss';
let Dashboard = Loadable({
   loader: () => import(/* webpackChunkName: "Home" */
'./pages/Dashboard'),
   loading: Loader,
});
const Login = Loadable({
   loader: () => import(/* webpackChunkName: "Login" */ './pages/Login'),
   loading: Loader,
});
Dashboard = Authorization(Dashboard);
const App = () => (
   <div>
      <AppLayout simplified={['/login']}>
         <Switch>
            <Route path="/" exact component={Dashboard} />
            <Route path="/login" exact component={Login} />
         </Switch>
      </AppLayout>
   </div>
);
export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { ConnectedRouter } from 'connected-react-router';
import {} from './redux';
import App from './App';
import { history, getStore } from './store';
ReactDOM.render(
   <Provider store={getStore()}>
      <ConnectedRouter history={history}>
         <App />
      </ConnectedRouter>
   </Provider>,
   document.getElementById('root')
);
```

```
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';
import { createBrowserHistory } from 'history';
import { routerMiddleware } from 'connected-react-router';
import { createStore, applyMiddleware } from 'redux';
import createRootReducer from './redux';
export const history = createBrowserHistory();
const middlewares = [thunk, routerMiddleware(history)];
```

```
const enhancers = [];
function configureStore(initialState) {
   const store = createStore(
      createRootReducer(history), // root reducer with router state
      initialState,
      composeWithDevTools(applyMiddleware(...middlewares), ...enhancers)
   );
   return store;
}
const store = configureStore({});
export function getStore() {
   return store;
}

import { combineReducers } from 'redux';
import { connectRouter } from 'connected-react-router';
import { userReducer } from './user';
import { authReducer } from './auth';
export default history =>
   combineReducers({
      router: connectRouter(history),
      user: userReducer,
      auth: authReducer,
   });

import { loginRequest } from '../api/users';
import { push } from 'connected-react-router';
import { handleReqError, showNotification } from
'../components/notifications';
export const SIGN_UP = 'SIGN_UP';
export const LOG_IN = 'LOG_IN';
export const LOG_OUT = 'LOG_OUT';
const QuornAuth = localStorage.getItem('QuornAuth');
const initialState = QuornAuth ? { QuornAuth } : null;
export const authReducer = (state = initialState, action) => {
   switch (action.type) {
      case SIGN_UP:
         localStorage.setItem('QuornAuth', action.payload.QuornAuth);
         return action.payload;
      case LOG_IN:
         localStorage.setItem('QuornAuth', action.payload.QuornAuth);
         localStorage.setItem('quorn_logged_in', true);
         return action.payload;
      case LOG_OUT:
         localStorage.removeItem('QuornAuth');
         localStorage.removeItem('quorn_logged_in');
         return null;
      default:
         return state;
   }
};
export const login = values => async dispatch => {
   try {
      const { data } = await loginRequest(values);
      console.log('auth handler: ', data);
      if (data.status === 'success') {
         dispatch({
            type: LOG_IN,
```

```
            payload: { QuornAuth: data.auth_token },
         });
      }
      showNotification({
         type: 'success',
         msg: 'Success',
         desc: data.message,
      });
      return data;
   } catch (err) {
      handleReqError(err);
      return false;
   }
};
export const logOut = (reload = true) => dispatch => {
   dispatch({ type: LOG_OUT });
   dispatch(push('/login'));
   if (reload) {
      window.location.reload();
   }
};

import React from 'react';
// import PropTypes, { array } from 'prop-types';
import { useSelector } from 'react-redux';
import { Redirect } from 'react-router-dom';
import { useLocation } from 'react-router-dom';
const getRedirectPath = location => {
   return location.pathname + location.search;
};
const Authorization = Component => {
   return function AuthorizedComponent(props) {
      const user = useSelector(state => state.user);
      const location = useLocation();
      if (!user) {
         // user in not logged in
         return (
            <Redirect
               to={{
                  pathname: '/login',
                  state: { redirectFrom: getRedirectPath(location) },
               }}
            />
         );
      }
      return <Component {...props} />;
   };
};
// Authorization.propTypes = {
//     allowedRoles: PropTypes.array.isRequired,
// };
export default Authorization;

import React, { useState } from 'react';
import Login from 'ant-design-pro/lib/Login';
import { Checkbox } from 'antd';
import { useDispatch } from 'react-redux';
import { push } from 'connected-react-router';
```

```
import { useLocation } from 'react-router-dom';
import Actions from '../redux/actions';
const { login } = Actions.auth;
const { setUser } = Actions.user;
const { Password, Submit, UserName } = Login;
const LoginPage = () => {
   const [autoLogin, setAutoLogin] = useState(false);
   const dispatch = useDispatch();
   const location = useLocation();
   return (
      <>
         <div className="login-wrap">
            <Login
               onSubmit={async (err, values) => {
                  const result = await dispatch(await login(values));
                  console.log(result);
                  if (result.status === 200) {
                     console.log('here');
                     dispatch(setUser(result.user, autoLogin));
                     if (location.state && location.state.redirectFrom) {
                        push(location.state.redirectFrom);
                     } else {
                        push('/');
                     }
                  }
               }}
            >
               <UserName placeholder="example@test.com" name="email" />
               <Password placeholder="password" name="password" />
               <div>
                  {/*TODO keep me logged in and reset password
functionality*/}
                  <Checkbox
                     checked={autoLogin}
                     onChange={e => {
                        setAutoLogin(e.target.checked);
                     }}
                  >
                     Keep me logged in
                  </Checkbox>
                  <a style={{ float: 'right' }} href="">
                     Forgot password
                  </a>
               </div>
               <Submit>Login</Submit>
               <div>
                  <a style={{ float: 'right' }} href="">
                     Register
                  </a>
               </div>
            </Login>
         </div>
      </>
   );
};
// Login.propTypes = {};
export default LoginPage;
```

```
import { notification } from 'antd';
/*
type:
  success
  error
  info
  warning
*/
export function showNotification({ type = 'info', msg, desc, duration = 0,
key }) {
   return notification[type.toLowerCase()]({
      message: msg,
      description: desc,
      duration,
      key,
   });
}
export function handleReqError(error) {
   console.log(error);
   if (error.response) {
      if (error.response.status === 404) {
         showNotification({
            type: 'error',
            msg: 'Error 404',
            desc: 'Not Found',
         });
      } else if (error.response.status === 403) {
         showNotification({
            type: 'error',
            msg: 'Error 403',
            desc: 'Forbidden',
         });
      } else if (error.response.status >= 400 && error.response.status <
500) {
         showNotification({
            type: 'error',
            msg: error.response.error,
            desc: 'Forbidden',
         });
      } else if (error.response.status >= 500) {
         showNotification({
            type: 'error',
            msg: 'Server error',
            desc: 'Something went wrong at the server side',
         });
      }
   }
}

import React from 'react';
import styled from 'styled-components';
import { ReactComponent as LoaderSvg } from '../assets/loader.svg';
const Bg = styled.div`
   position: fixed;
   z-index: 10000;
   background-color: transparent;
   left: 0;
   top: 0;
```

```
   right: 0;
   bottom: 0;
   display: flex;
   align-items: center;
   justify-content: center;
`;
// TODO: better error display
export const Loader = ({ error, pastDelay }) => {
   if (error) {
      console.log(error);
      return <div>Oh no, something went wrong!</div>;
   } else if (pastDelay) {
      return (
         <Bg>
            <LoaderSvg />
         </Bg>
      );
   } else {
      return null;
   }
};
```

Фрагменти коду серверної частини.

```
from mongoengine import *
class UserDocument (Document):
    meta = {
        'collection': 'users'
    }
    email = EmailField(unique=True)
    password = StringField(max_length=500, required=True)
    name = StringField(max_length=200, required=True)
    surname = StringField(max_length=200, required=True)
    ethereum_address = StringField(max_length=100, required=True,
unique=True)

import codecs
import hashlib
import datetime
import jwt
import json
from credentials import SALT
from mongo_models import UserDocument
from .Response import ResponseData
class User:
    def __init__(self):
        pass
    @staticmethod
    def create_user(user_data) -> ResponseData:
        try:
            user = UserDocument(
                email=user_data['email'],
                password=User.hash_password(user_data['password']),
                name=user_data['name'],
                surname=user_data['surname'],
                ethereum_address=user_data['ethereum_address'],
            )
            user.save()
            return ResponseData(200, {'created': True})
```

```
        except Exception as e:
            return ResponseData(400, {"error": str(e)})
    @staticmethod
    def get_user_by_email(email) -> UserDocument:
        user = UserDocument.objects(email=email)[0] or {}
        return user
    @staticmethod
    def login(user_data) -> ResponseData:
        user =
json.loads(User.get_user_by_email(user_data['email']).to_json())
        print('fetched: ', user)
        try:
            if not user:
                raise Exception('No such user')
            if user['password'] !=
User.hash_password(user_data['password']):
                raise Exception('Wrong password')
            auth_token = User.encode_auth_token(user['email'])
            if auth_token:
                return ResponseData(200, {
                    'status': 'success',
                    'message': 'Successfully logged in.',
                    'auth_token': auth_token.decode(),
                    'user': user
                })
        except Exception as e:
            print(e)
            return ResponseData(400, {
                'error': str(e),
            })
    #
    # service methods
    @staticmethod
    def encode_auth_token(email) -> jwt:
        try:
            payload = {
                'exp': datetime.datetime.utcnow() +
datetime.timedelta(days=1),
                'iat': datetime.datetime.utcnow(),
                'sub': email
            }
            return jwt.encode(
                payload,
                SALT,
                algorithm='HS256'
            )
        except Exception as e:
            raise e
    @staticmethod
    def decode_auth_token(auth_token):
        try:
            payload = jwt.decode(auth_token, SALT)
            return payload['sub']
        except jwt.ExpiredSignatureError:
            raise Exception('Signature expired. Please log in again.')
        except jwt.InvalidTokenError:
            raise Exception('Invalid token. Please log in again.')
    @staticmethod
    def hash_password(password):
```

```
        res = hashlib.sha256(codecs.encode(password, 'ascii')).hexdigest()
        return res

import json
from flask import Response
class ResponseData:
    def __init__(self, code: int, result: dict):
        self.code = code
        self.result = result
    def to_flask_response_json(self) -> Response:
        return Response(
            response=json.dumps(self.result),
            status=self.code,
            mimetype='application/json'
        )
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ВЕБ-ДОДАТОК ДЛЯ ОРГАНІЗАЦІЇ ОНЛАЙН-ЧЕРГИ З ПІДТРИМКОЮ МОЖЛИВОСТІ ОЦІНЮВАННЯ ЯКОСТІ ПОСЛУГ

Виконала: Корунська Анна Михайлівна

Керівник: доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна

Київ – 2020

ІДЕЯ

Ідея проекту полягає в розробленні веб-додатку, який вирішував би проблеми довіри користувачів до відгуків про роботу спеціалістів, що можна знайти на онлайн-сервісах.

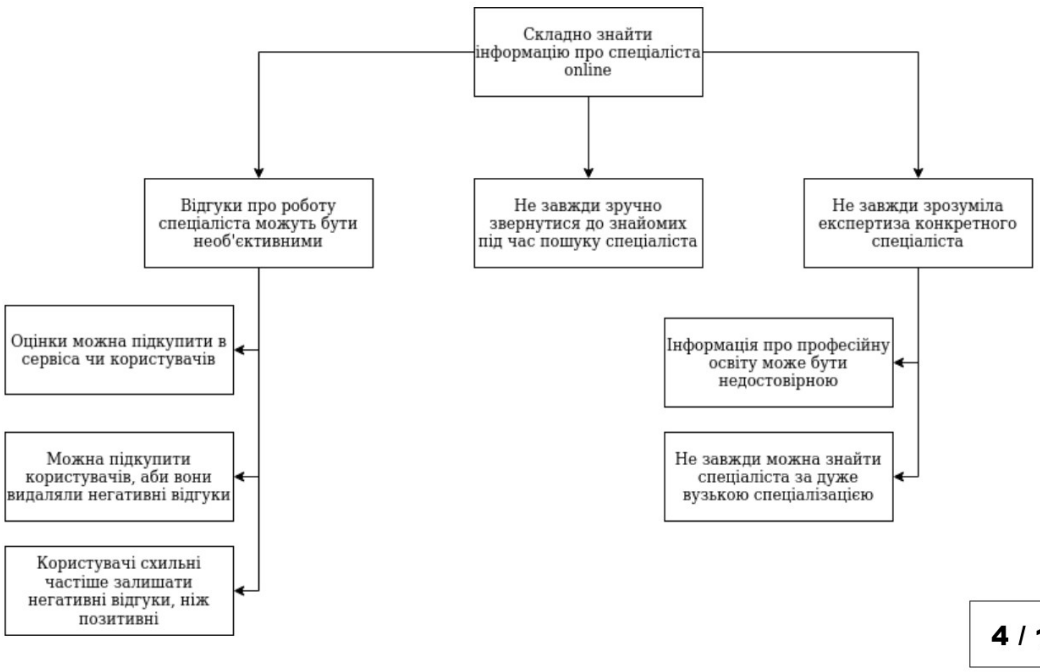
МЕТА І ЗАВДАННЯ ПРОЄКТУ

Мета: підвищення довіри користувачів онлайн-черг до відгуків та оцінок про роботу закладів чи окремих спеціалістів шляхом розроблення нового веб-сервісу, що гарантує неможливість шахрайства з даними чи їх цензурування.

Завдання проєкту:

1. Проаналізувати системи онлайн-черг
2. Розробити веб-додаток онлайн-черги, яка б забезпечила захищене від шахраювання зберігання даних
3. Протестувати веб-додаток
4. Описати подальші шляхи розвитку веб-додатку

ДЕРЕВО ПРОБЛЕМ



АКТУАЛЬНІСТЬ

Відсутність довіри до рейтингових даних:

- Підкуп оцінок у сервісу чи користувачів
- Видалення негативних коментарів сервісом
- Цензурування

АНАЛОГИ

helsi

DOC UA

головний сервісний центр
МВС України

ВИКОРИСТАННЯ ТЕХНОЛОГІЇ БЛОКЧЕЙН

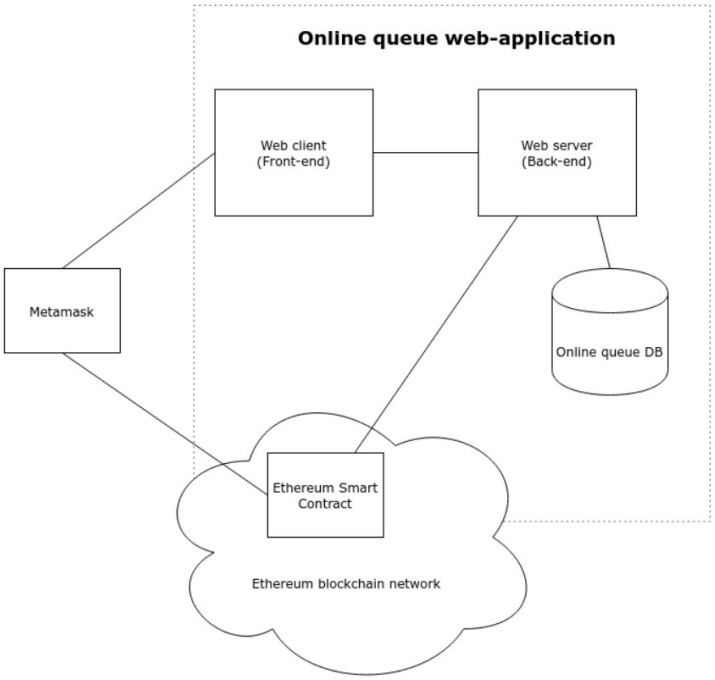
Технологія блокчейн забезпечує розроблюваному ПЗ:

- цілісність історії змін бази даних;
- можливість проведення аудиту в режимі реального часу;
- фіксацію даних, що включаються до бази даних, у часі;
- відсутність необхідності довіри йому.

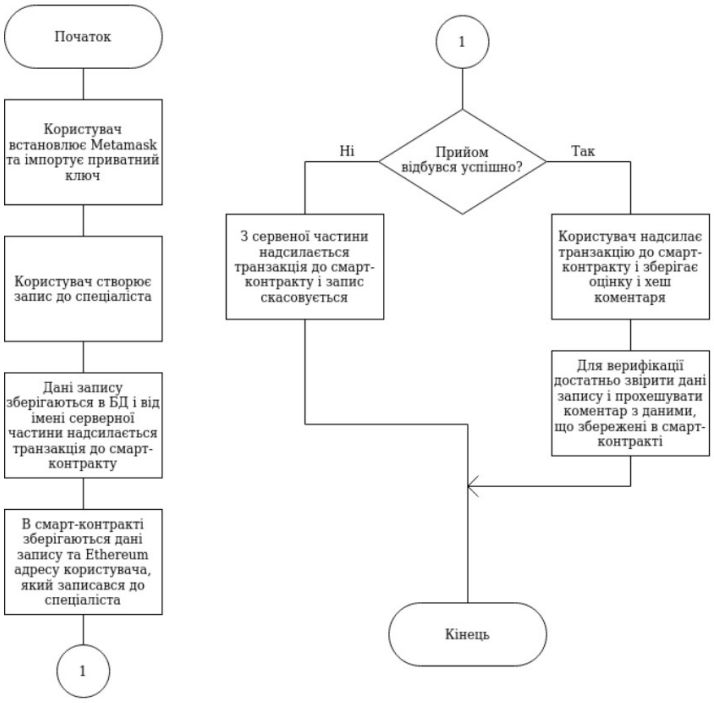
Прозорість побудови рейтингу та неможливість підробки коментарів в розробленому веб-додатку є можливою завдяки використанню смарт-контрактів.

ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ

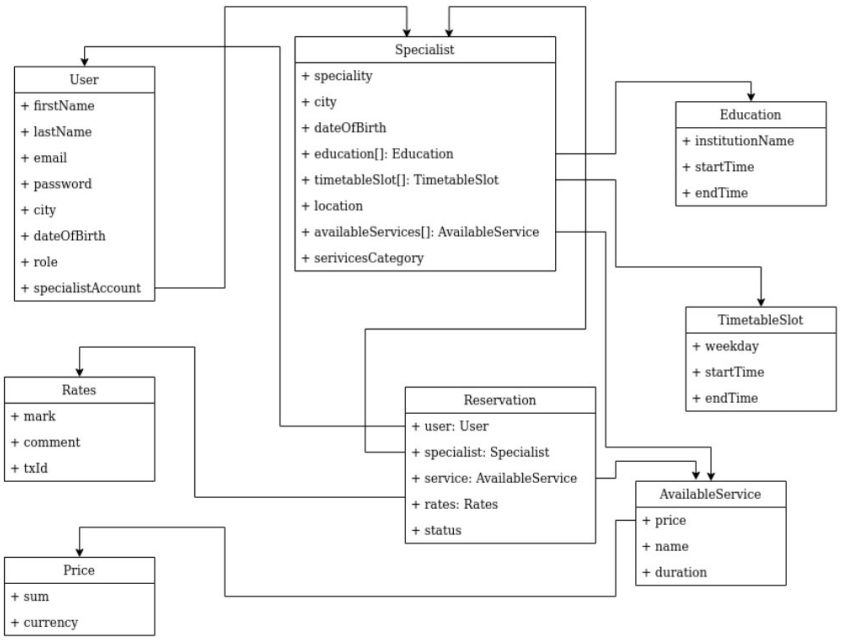
УЗАГАЛЬНЕНА СТРУКТУРНА ОРГАНІЗАЦІЯ ВЕБ-ДОДАТКУ



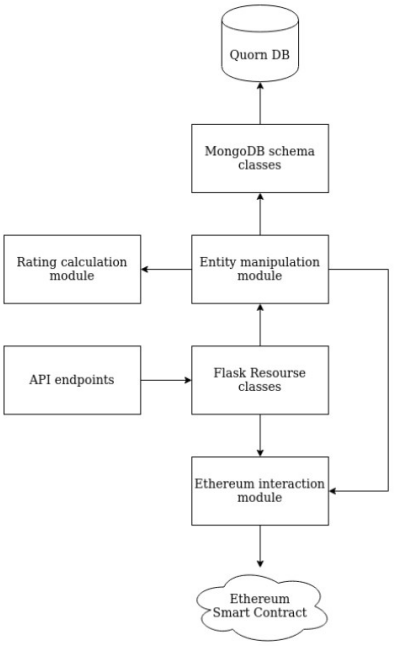
ВЗАЄМОДІЯ СИСТЕМИ ЗІ СМАРТ-КОНТРАКТОМ. БЛОК-СХЕМА АЛГОРИТМУ



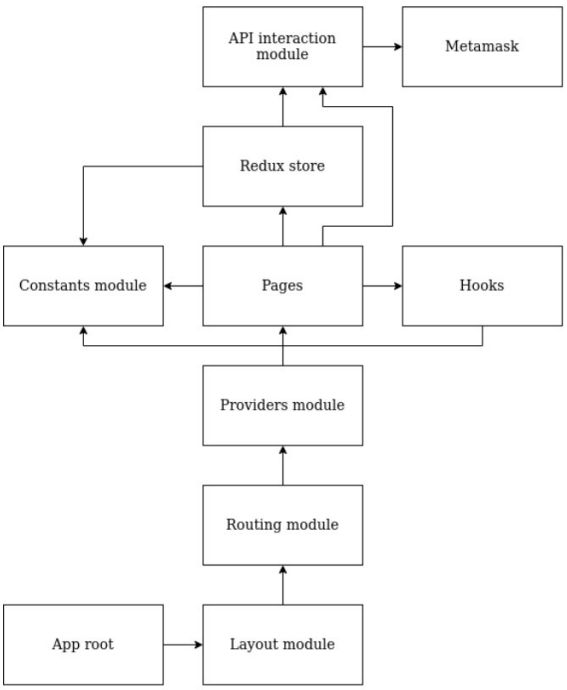
ERD-ДІАГРАМА БАЗИ ДАНИХ СИСТЕМИ



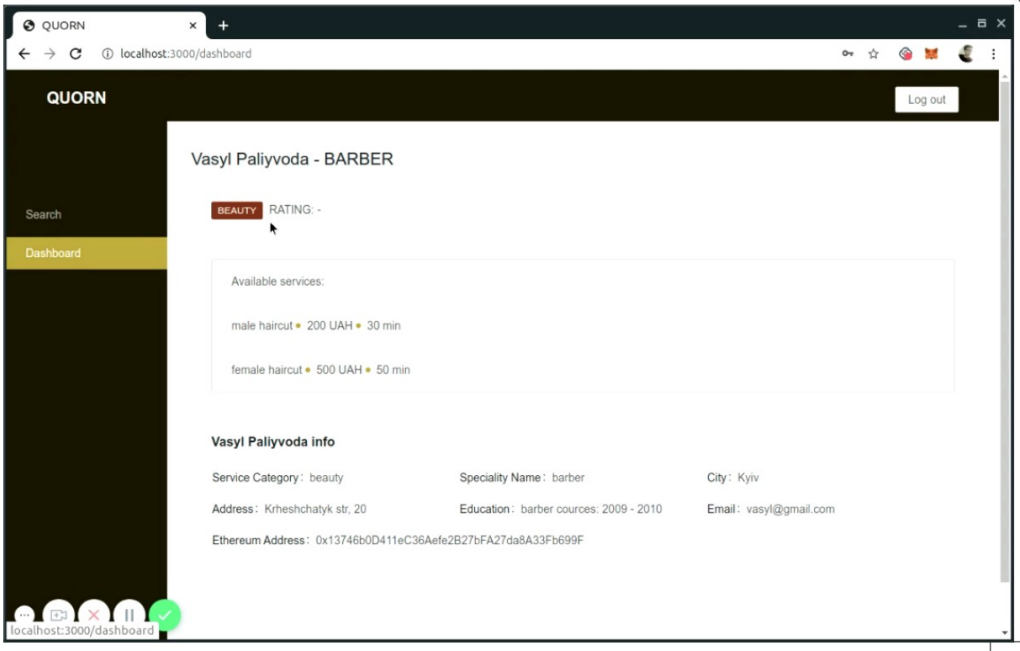
ДІАГРАМА МОДУЛІВ СЕРВЕРНОЇ ЧАСТИНИ



ДІАГРАМА МОДУЛІВ КЛІЄНТСЬКОЇ ЧАСТИНИ



ДЕМОНСТРАЦІЯ ВЕБ-ДОДАТКУ



АПРОБАЦІЯ

1. V Міжнародна науково-практична конференція «Інформаційні технології в освіті, науці і техніці» (ІТОНТ-2020). Результати опубліковані в вигляді тез доповіді.
2. Підготовлено пакет документів на отримання авторського свідоцтва.

ВИСНОВКИ

1. Проаналізовано проблеми сервісів онлайн-черг.
2. Проаналізовано існуючі програмні рішення.
3. Обрано іструменти розробки та обгрунтовано доцільність використання технології блокчейн на смарт-контрактів.
4. Розроблено та наведено структуру веб-додатку, діаграми модулів кожного компонента системи, а також діаграму бази даних.
5. Розроблено веб-додаток, який успішно забезпечує протидію цензуруванню чи спотворенню оцінок чи коментарів.
6. Протестовано веб-додаток та перевірено відповідність вимогам, що були сформульовані перед розробленням.
7. Запропоновано подальші шляхи розвитку веб-додатку.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ ОРГАНІЗАЦІЇ ОНЛАЙН-ЧЕРГИ З ПІДТРИМКОЮ МОЖЛИВОСТІ ОЦІНЮВАННЯ ЯКОСТІ ПОСЛУГ

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проекту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анна КОРУНСЬКА

2019

ЗМІСТ

1. Об’єкт випробувань……...3
2. Мета тестування…….3
3. Методи тестування…….……3
4. Засоби та порядок тестування…….……..4

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Сервіс онлайн черги, що являє собою веб-додаток, серверна частина якого реалізована мовою програмування Python, а клієнтська за допомогою бібліотеки React, а також смарт-контракти, написані мовою Solidity, що опубліковані в блокчейн-мережі Ethereum.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1. Функціональна працездатність елементів сторінок веб-додатку.
2. Відповідність функціональності веб-сервісу вимогам Технічного завдання.
3. Відповідність фукнцкіональності смарт-контрактів вимогам Технічного завдання.
4. Забезпечення належного рівня безпеки даних.
5. Зручність роботи з веб-додатком;

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1. Функціональне тестування, зокрема на рівні Critical path test (базове тестування).
2. Тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування).
3. Тестування фукнціональності смарт-контрактів за допомогою unit-тестів.
4. Тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію SpecFlow.

Працездатність веб-додатку перевіряється шляхом:

1. Динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати.
2. Динамічного ручного тестування на відповідність функціональним вимогам.
3. Статичного тестування коду.
4. Тестування веб-додатку в різних web-браузерах.
5. Тестування при максимальному навантаженні.
6. Тестування стабільності роботи при різних умовах.
7. Тестування зручності використання.
8. Тестування інтерфейсу.
9. Тестування смарт-контрактів.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ ОРГАНІЗАЦІЇ ОНЛАЙН-ЧЕРГИ З ПІДТРИМКОЮ МОЖЛИВОСТІ ОЦІНЮВАННЯ ЯКОСТІ ПОСЛУГ

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проекту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анна КОРУНСЬКА

2020

ЗМІСТ

1. Опис структури веб-додатку.....3
2. Процедура авторизації користувача.....4
3. Процедура реєстрації користувача.....6
4. Пошук спеціалістів.....8
5. Сторінка спеціаліста.....10
6. Особистий кабінет.....12

1. Опис структури веб-додатку

Розроблений веб-додаток складається із серверної та клієнтської часитини, а також смарт-контрактів. Напряму користувачі взаємодіють із клієнтською частиною, тобто веб-сторінками, через браузер.

Сторінки відображаються динамічно і залежать від ролі користувача, а також від того, чи є користувач авторизованим.

2. Процедура авторизації користувача

При переході на сторінку веб-сервісу, неавторизований користувач опиняється на сторінці авторизації (рис.1) за адресою /login.

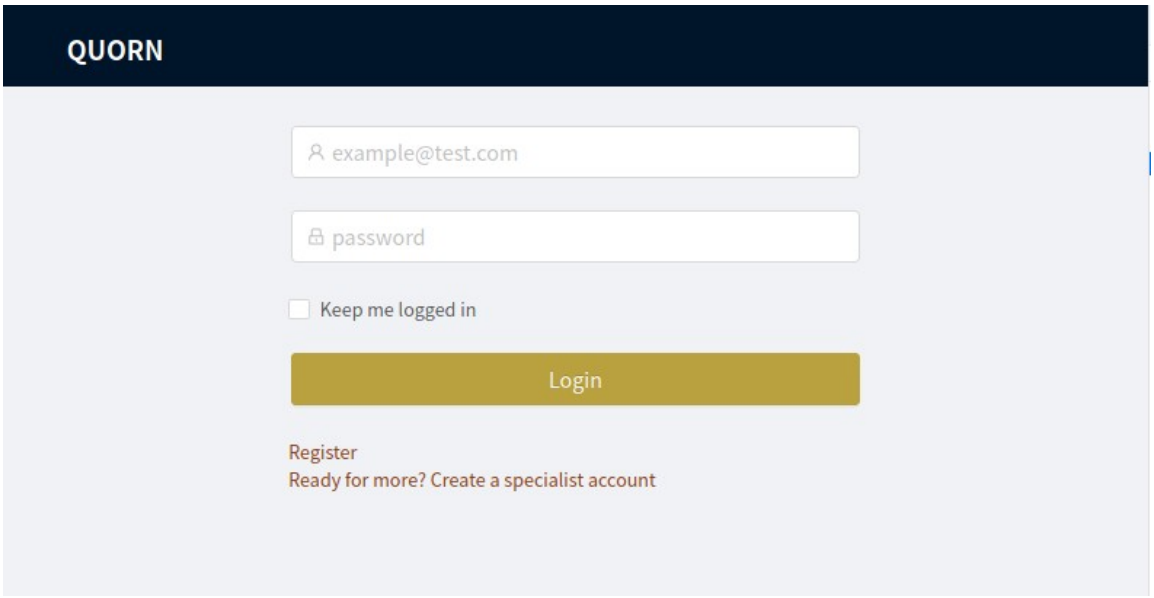


Рис.1. Сторінка авторизації

На цій сторінці користувач повинен ввести свої логін та пароль. Поля динамічно перевіряються на коректність ввводу і користувач отримує повідомлення у разі неправильно введених значень (рис.2).

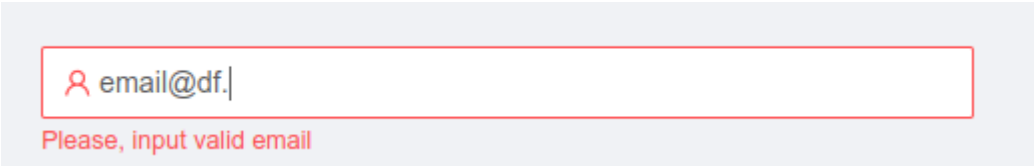


Рис.2. Приклад повідомлення про некоректне значення елекнронної пошти

Сторінка також містить посилання на сторінки реєстрації для користувача та спеціаліста, а також поле типу checkbox, завдяки якому

користувач може обрати, чи бажає він залишатися авторизованим у системі після того, як буде закрита поточна вкладка веб-браузера.

У випадку, якщо користувач ввів логін або пароль неправильно, він отримає повідомлення про помилку (рис.3).

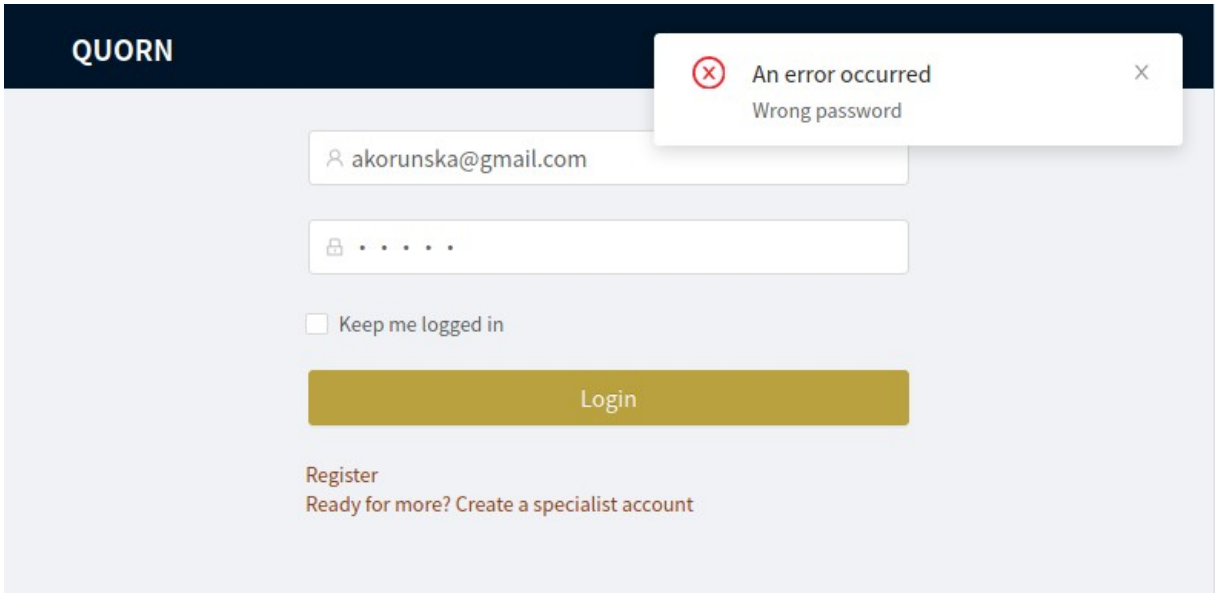


Рис.3. Повідомлення про невірно введений пароль під час авторизації

Якщо користувач ввів пароль вірно, він отримає відповідне повідомлення про успішну авторизацію (рис.4) і буде перенаправлений на сторінку пошуку спеціалістів.

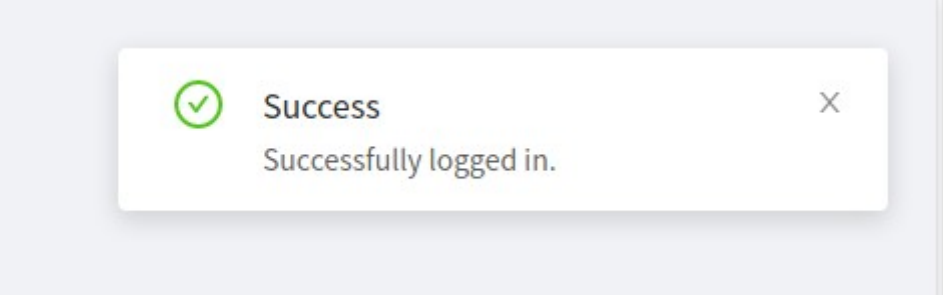


Рис.4. Повідомлення про успішну авторизацію

3. Процедура реєстрації користувача

Користувач може перейти на сторінку реєстрації (рис. 5), яка знаходиться за адресою /register, або ж на сторінку реєстрації спеціалістів, що знаходиться за адресою /register-specialist.

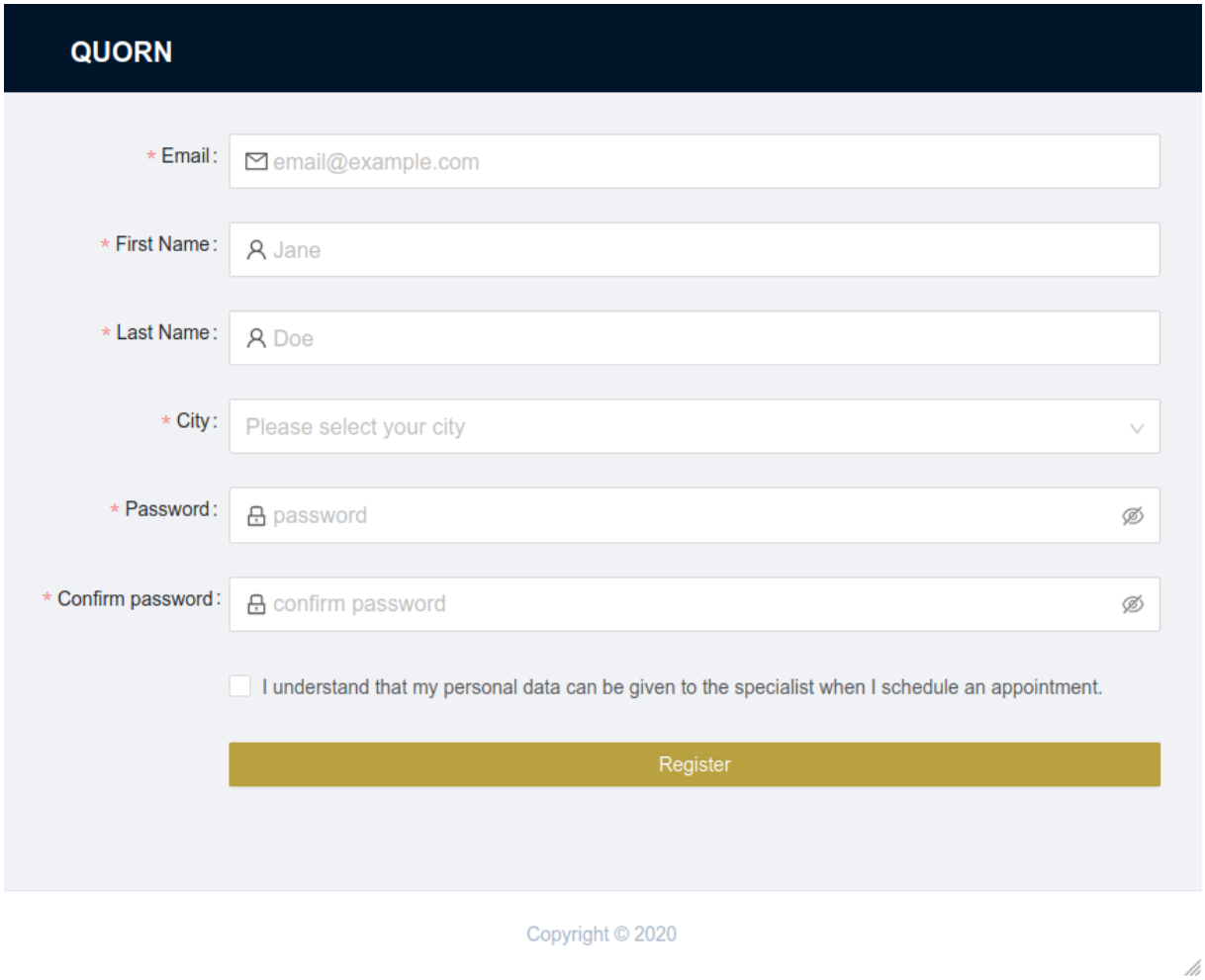


Рис. 5 Сторінка реєстрації

Сторінка містить обов'язкові поля, які користувачу потрібно заповнити для того, щоб успішно створити обліковий запис, тобто електронну адресу, місто, ім'я, прізвище, тощо.

Користувач повинен також ввести свій пароль двічі, для того, аби не втатити доступ до облікового запису через помилковий пароль при реєстрації.

У разі, якщо введені паролі не співпадають, користувач отримає відповідне повідомлення (рис.6).

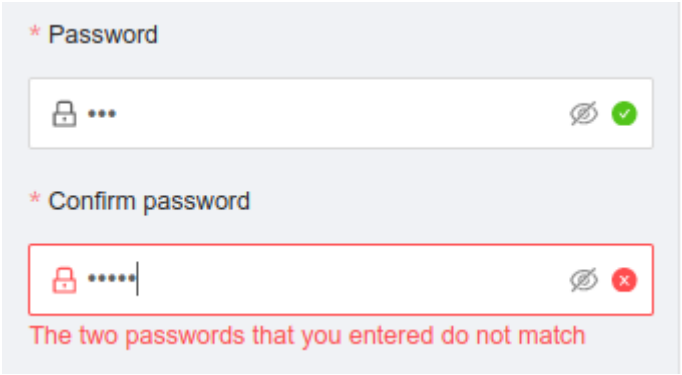


Рис. 6. Повідомлення про помилку у разі, якщо введені при реєстрації паролі не співпадають

Також, реєстрацію неможливо завершити, доки користувач не дозволить передавати обмежену кількість персональних даних спеціалістам при записі до них у чергу (рис. 7).

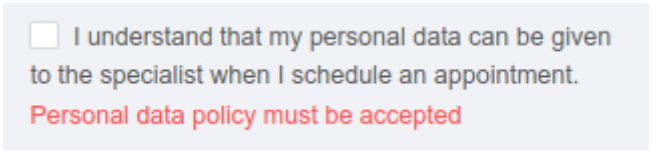


Рис. 7 . Повідомлення про необхідність погодитись із дозволом на передачу персональних даних

4. Пошук спеціалістів

Після успішної авторизації користувач потрапляє на основну частину веб-сервісу. За замовчуванням, він перенаправляється на сторінку пошуку спеціалістів (рис. 8) за адресою /login.

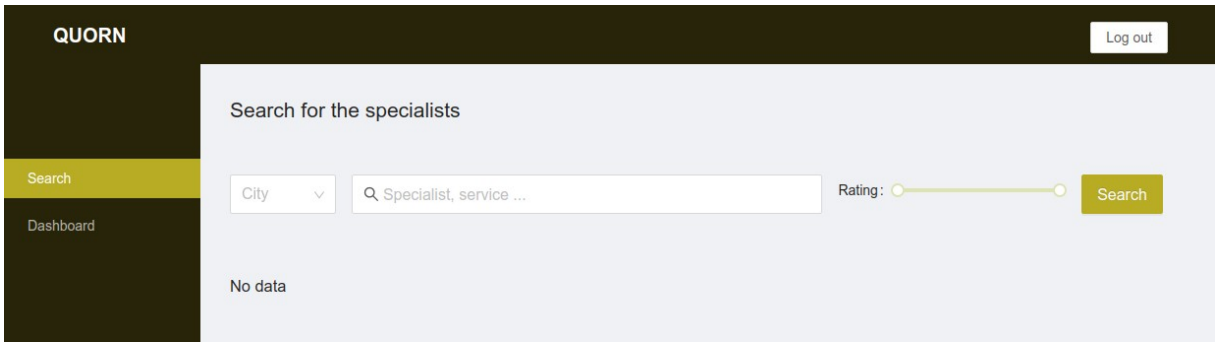


Рис. 8. Сторінка пошуку спеціалістів

Користувач обирає необхідні параметри пошуку. Після завантаження (рис 9) користувач може побачити результати пошуку (рис. 10).

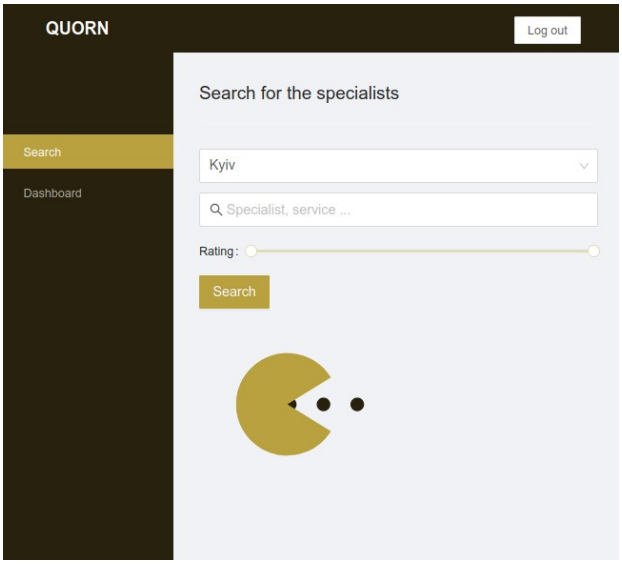


Рис. 9 Заватнаження результатів

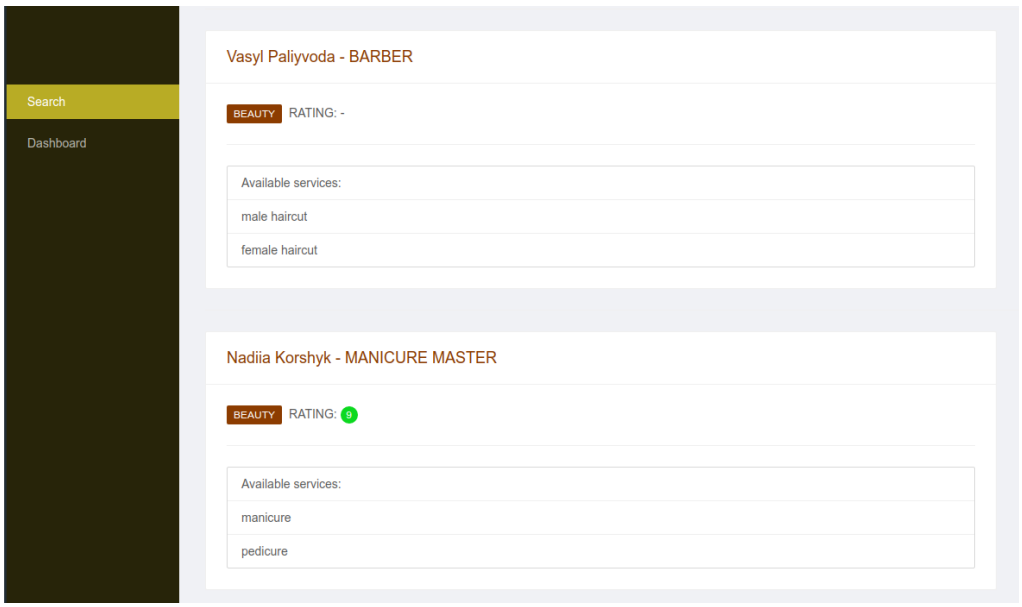


Рис. 10 Результати пошуку

5. Сторінка спеціаліста

Зі сторінки результатів пошуку користувач потрапляє на сторінку спеціаліста за адресою /specialist/:specialistId.

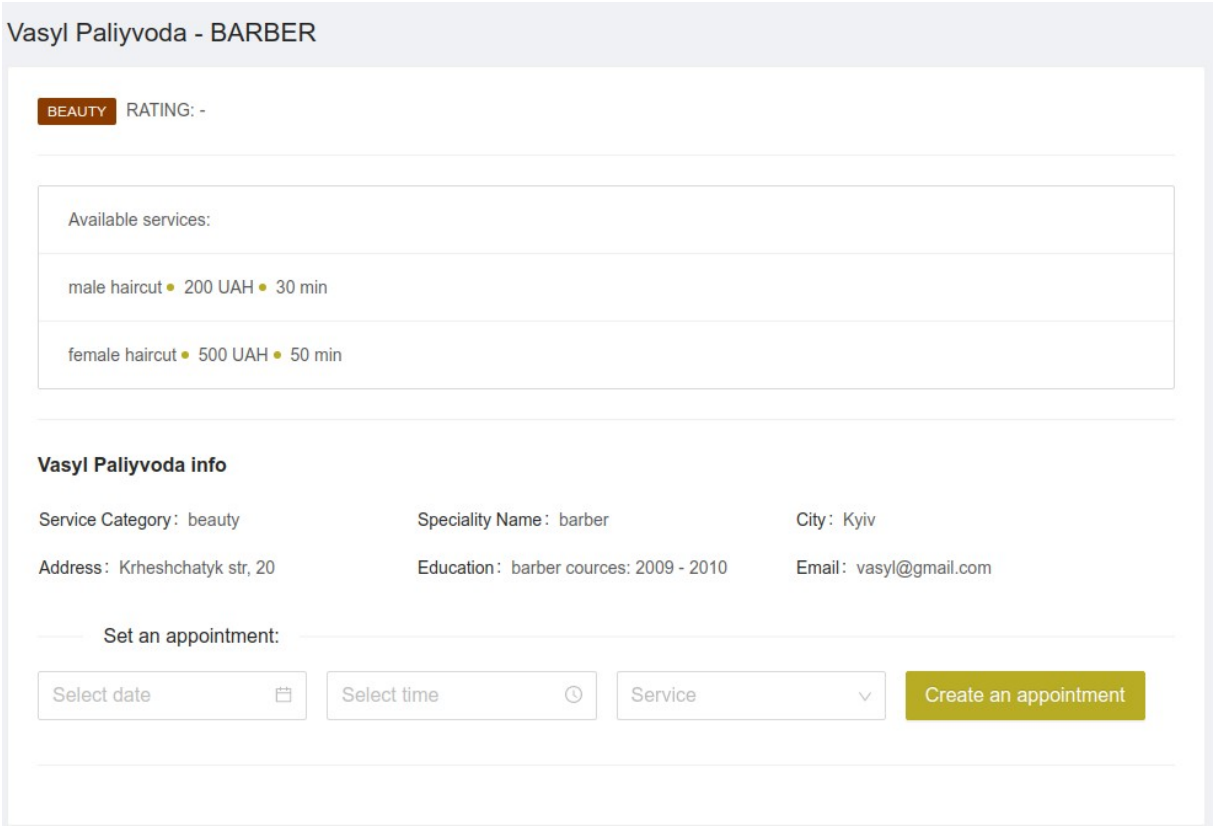


Рис. 11 Сторінка спеціаліста

На сторінці спеціаліста користувач может переглянути детальну інформацію про спеціаліста за записатися до нього на обрану послугу. Якщо запис виконано успішно, користувачу буде відображено відповідне повідомлення.

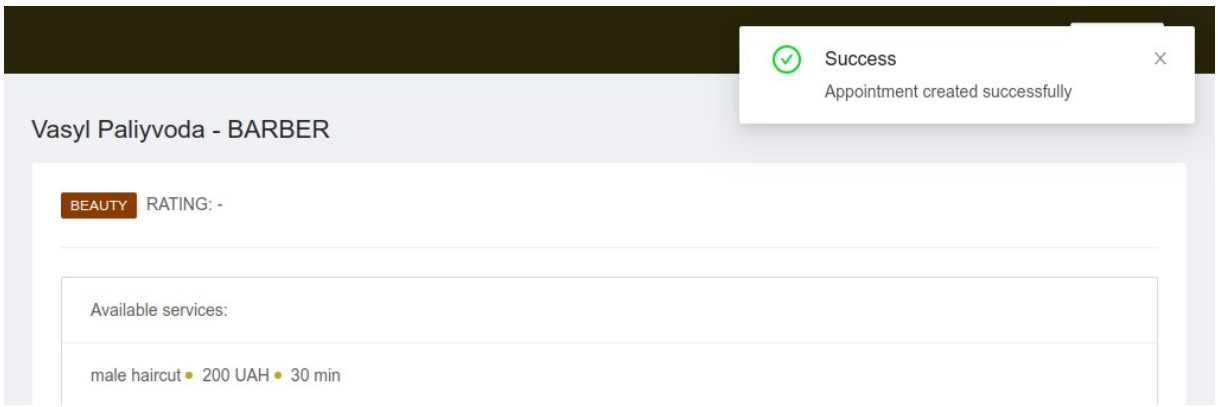


Рис.12 Повідомлення про успішну реєстрацію у черзі

6. Особистий кабінет

І користувачу, і спеціалісту доступний особистий кабінет за адресою /dashboard.

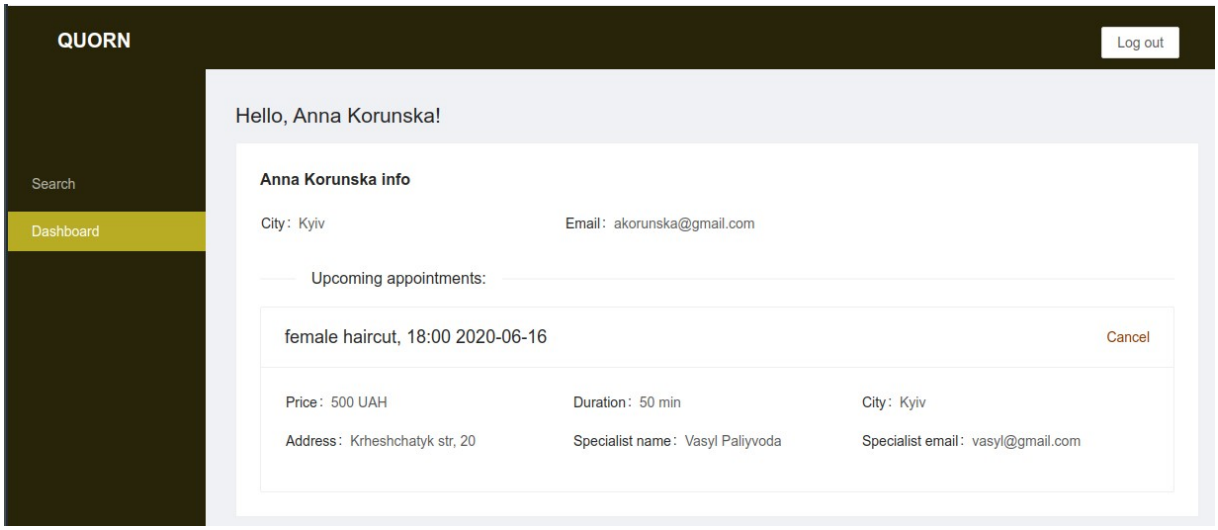


Рис. 13 Особистий кабінет користувача

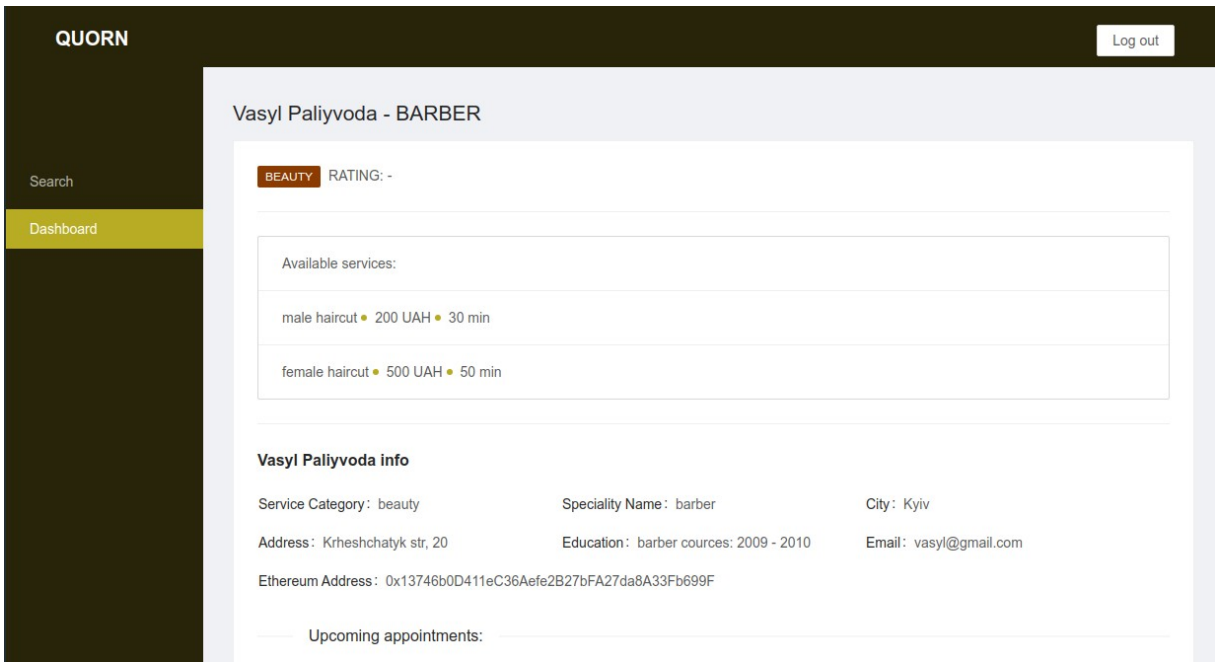


Рис.14 Особистий кабінет спеціаліста

В особистому кабінеті користувачі та спеціалісти можуть переглядати інформацію про себе та свої записи у чергу.

Кожен запис у черзі відображається детально і до моменту його настання у спеціаліста та у користувача є можливість його скасувати. Після того, у користувача є можливість поставити відгук та оцінку відвідуванню.